Eventually Safe Languages

Simon Iosti, Denis Kuperberg

Eventually Safe Languages

Simon Iosti[1] and Denis Kuperberg[2]

[1] Verimag, Université Grenoble-Alpes, France
[2] CNRS, LIP, ENS Lyon, France

Abstract. Good-for-Games (GFG) automata constitute a sound alternative to determinism as a way to model specifications in the Church synthesis problem. Typically, inputs for the synthesis problem are in the form of LTL formulas. However, the only known examples where GFG automata present an exponential gap in succinctness compared to deterministic ones are not LTL-definable. We show that GFG automata still enjoy exponential succinctness for LTL-definable languages. We introduce a class of properties called "eventually safe" together with a specification language $E\nu$TL for this class. We finally give an algorithm to produce a Good-for-Games automaton from any $E\nu$TL formula, thereby allowing synthesis for eventually safe properties.

1 Introduction

Synthesis is one of the most classical applications of automata theory. It asks, given a specification, whether there exists a reactive system complying with it. We also want to automatically build such a system when it exists. The specification is typically given in a logic such as Linear Temporal Logic (LTL). The problem was solved positively by Büchi and Landweber [5] for the case of ω-regular specifications. The usual approach to this problem consists in building a deterministic automaton from the specification, and then solving a game based on this automaton. Henzinger and Piterman [11] have proposed a model of Good-For-Games (GFG) automata as a weakening of determinism that is still sound for solving the synthesis problem. An automaton is GFG if there exists a strategy that resolves the non-deterministic choices, by taking into account only the prefix of the input ω-word read so far. The strategy must guarantee to build an accepting run whenever the input word is in the language of the automaton. In [15], the question of succinctness of GFG automata compared to deterministic ones is answered. A family (L_n) of languages is exhibited, such that for each n there is a GFG coBüchi automaton of size n for L_n, but any deterministic Streett automaton for L_n must have size exponential in n. Therefore, GFG automata offer a promising alternative to deterministic ones for synthesis, and this work is part of an effort to systematically study their applicability in this context.

However, one of the potential issues with the use of GFG automata for synthesis lies in the fact that the most usual specification formalism for synthesis is LTL. It is therefore natural to ask whether GFG automata can be useful in this context. We show that the languages L_n witnessing succinctness of GFG

automata are not LTL-definable. Moreover, a close look at the structure of GFG coBüchi automata, as studied in [15], suggests that the ability to permute states is an essential feature of non-trivial GFG automata. It is therefore plausible that GFG automata are no longer succinct (compared to deterministic ones) for LTL-definable languages, where such permutations are forbidden [7].

We answer this question here, by building a family (K_n) of LTL-definable languages presenting the same succinctness gap as the family (L_n) between GFG and deterministic automata. Although this shows that GFG automata are still succinct for LTL-definable languages, the issue of practicability is still unclear, due to the fact that the LTL formulas representing K_n have exponential size. Moreover, we show that there are simple μ-calculus formulas of linear size for the same languages. Interestingly, a by-product of this work is the exhibition of the family K_n as a candidate witness for an exponential gap succinctness of linear μ-calculus compared to LTL, a problem that is open to our knowledge. This suggests that μ-calculus is more suited than LTL for describing specifications that are recognized by small GFG automata. We therefore aim at proposing a framework based on μ-calculus for building succinct GFG automata.

This leads us to a second issue standing in the way of bringing GFG automata to practical applications. Due to their semantic definition, building GFG automata is a hard problem and requires an understanding of their syntactical shape. A first way to achieve this has been given in [14], building GFG automata in an incremental way from non-deterministic ones. The algorithm tries bigger and bigger automata until a GFG one is reached, the worst case being when a full determinization construction is needed. The only knowledge about GFG automata that is used in this construction is in the subroutine used to test whether an automaton is GFG. We propose here an alternative approach, building automata that are GFG by construction, using the understanding acquired in [15] about the structure of GFG coBüchi automata.

Considering restricted classes of specifications is a classical way to try to tackle the difficulty of the synthesis problem. The classes of safety and liveness properties [1] have gathered particular interest [19, 21], as they simplify algorithms while expressing typical requirements on reactive systems. We introduce a class of properties called "eventually safe" and noted *ESafe*, for which we give a specification language $E\nu$TL and an algorithm systematically producing GFG automata from this language. The class *ESafe* can be seen as a natural compromise between safety and liveness, and is defined as the class of languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language. Equivalently, the class *ESafe* is the class of prefix-independent coBüchi languages. As an example, the following specification can be formalized in *ESafe*: "after being started, the system must eventually start interacting with external agents, and must answer their requests within a fixed finite time".

Both families (L_n) and (K_n) are expressible in the logic $E\nu$TL in a very natural way and with formulas of size linear in n. This approach is orthogonal to the one from [14] that we outlined above. Here, we restrict the class of inputs to the class *ESafe* that is natural for verification purposes, and for which GFG

automata are well-suited. We show that unfortunately, translating a formula of $E\nu$TL to a GFG automaton is still doubly exponential in the worst case. This is not surprising, as it was shown in [4] that this is already the case for translating LTL (or linear μ-calculus) formulas for "bounded" languages, i.e. languages that are both safe and co-safe, to GFG automata. However we believe that this model is worth exploring, in order to understand the power and possible limitations of GFG automata for synthesis. Moreover, recent works such as [8, 13] rely on a modular treatment of specifications, and can call subroutines to build automata for restricted fragments of LTL. This is particularly suitable to embed GFG automata for well-behaved fragments, and the present work brings a clearer understanding of the possibilities and theoretical limitations of this approach.

Let us give another example of application of the $E\nu$TL formalism, in the spirit of this modular approach. Properties required of systems are often subject to fairness assumptions. This is expressed by specifications of the form $\psi \Rightarrow \varphi$ for some fairness assumption ψ, that typically consists in liveness properties. This can be treated by building a GFG automaton for $\neg\psi$, in addition to the automaton for φ. In this context, $E\nu$TL would for example allow to model the fairness assumption that a finite set of agents (that can be dynamically renamed) will all be activated infinitely many times, as the complement of such a language is similar to the language L_n discussed in this work.

Outline of the paper

We start by recalling definitions on logical formalisms and automata in Section 2. In Section 3, we recall the definition of the (L_n) language family from [15], and show that it is not LTL-definable. In Section 4, we define the (K_n) family, show that it also witnesses succinctness of GFG automata compared to deterministic ones, and give a family of LTL formulas of exponential size for the languages (K_n). In Section 5, we define the safety fragment $S\nu$TL of linear μ-calculus, show that it is equivalent to safety languages, and build the logic $E\nu$TL based on its syntax by using an alternative semantic. We show that both families of languages (K_n) and (L_n) have linear-size representations in $E\nu$TL, and give a generic algorithm to translate an $E\nu$TL formula to a GFG coBüchi automaton. We also exhibit an $E\nu$TL formula witnessing that the translation to GFG coBüchi automata is doubly exponential in the worst case. Detailed proofs can be found in Appendix.

2 Definitions

We will use Σ to denote an arbitrary finite alphabet. The empty word is denoted ε. If $i \leq j$, the set $\{i, i+1, i+2, \ldots, j\}$ is denoted $[i, j]$. The set of finite words on Σ is denoted Σ^*, and the set of infinite words Σ^ω. We note $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. If $X \subseteq \Sigma^\omega$, we note $\mathit{pref}(X)$ the set of finite prefixes of words in X, and $\mathit{suff}(X)$ the set of infinite suffixes of words in X. A set $X \subseteq \Sigma^\omega$ is *prefix-independent* if for all $u, v \in \Sigma^*$ and $w \in \Sigma^\omega$, we have $uw \in X \Leftrightarrow vw \in X$. A set $X \subseteq \Sigma^\omega$ is *suffix-closed* if $\mathit{suff}(X) = X$.

2.1 Logic

We define here the linear temporal logic (LTL) and the linear μ-calculus.

The syntax of LTL is defined with the following grammar, where a ranges over Σ:

$$\varphi := a \mid \varphi \vee \varphi \mid \neg\varphi \mid \odot\varphi \mid \varphi \mathbf{U} \varphi$$

The semantic $[\![\varphi]\!] \subseteq \Sigma^\omega$ of a formula φ of LTL is defined recursively on the formula:

- $[\![a]\!] = \{aw \mid w \in \Sigma^\omega\}$,
- $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$,
- $[\![\neg\varphi]\!] = \Sigma^\omega \setminus [\![\varphi]\!]$,
- $[\![\odot\varphi]\!] = \{aw \in \Sigma^\omega \mid a \in \Sigma, w \in [\![\varphi]\!]\}$,
- $[\![\varphi \mathbf{U} \psi]\!] = \{a_0 a_1 \cdots \in \Sigma^\omega \mid \exists i \in \mathbb{N}, \forall j < i, a_j a_{j+1} \cdots \in [\![\varphi]\!], \text{ and } a_i a_{i+1} \cdots \in [\![\psi]\!]\}$.

Let a be an arbitrary letter in Σ and φ, ψ be LTL formulas. We will use the syntactic sugar $\varphi \wedge \psi$, $\top$, $\bot$, $\mathbf{F}\varphi$, $\mathbf{G}\varphi$ and $\varphi \mathbf{W} \mathbf{U} \psi$ as shorthands for $\neg((\neg\varphi) \vee (\neg\psi))$, $a \vee \neg a$, $\neg\top$, $\top \mathbf{U} \varphi$, $\neg\mathbf{F}\neg\varphi$ and $\mathbf{G}\varphi \vee \varphi \mathbf{U} \psi$ respectively.

The linear μ-calculus has the following syntax, where a ranges over Σ, and X over a countable set V of variables:

$$\varphi := a \mid X \mid \varphi \vee \varphi \mid \neg\varphi \mid \odot\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

Its semantic $[\![\varphi]\!]_{\mu,val}$ relative to a valuation $val : V \to 2^{\Sigma^\omega}$ of the variables is defined similarly to the semantic of LTL for their common symbols (using the μ-calculus semantic instead of the LTL semantic), with the following additional rules for the new symbols, where *gfp* and *lfp* denote the greatest fixed point and the least fixed point operators respectively, and $val[X \to S]$ is the valuation val except for $val(X) = S$:

- $[\![X]\!]_{\mu,val} = val(X)$;
- $[\![\mu X.\varphi]\!]_{\mu,val} = \mathit{lfp}(S \to [\![\varphi]\!]_{\mu,val[X \to S]})$;
- $[\![\nu X.\varphi]\!]_{\mu,val} = \mathit{gfp}(S \to [\![\varphi]\!]_{\mu,val[X \to S]})$.

The semantic $[\![\varphi]\!]_\mu$ of a closed formula φ is $[\![\varphi]\!]_{\mu,\emptyset}$ where $\emptyset$ is the empty valuation.

The *DAG-size* of a formula is a measure of the size of the formula using the directed acyclic graph (DAG) representing the formula instead of the syntactic tree. We define formally the DAG-size $|\varphi|_{dag}$ of a formula φ as the size of the set $sub(\varphi)$ of subformulas of φ. This representation of a formula as a DAG is usually the one used in algorithms taking as input LTL or μ-calculus formulas (e.g. the translation from a LTL formula to a Büchi automaton), and is therefore a more sensible measure of the size of a formula than the size of its syntactic tree.

2.2 Automata

A non-deterministic automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, q_0, \Delta, F)$ where Q is the set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\Delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. If for all $(p, a) \in Q \times \Sigma$ there is at most one $q \in Q$ such that $q \in \Delta(p, a)$, we say that $\mathcal{A}$ is *deterministic*.

If $u = a_1 \dots a_n$ is a finite word of Σ^*, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 \dots q_n$ such that for all $i \in [1, n]$, we have $q_i \in \Delta(q_{i-1}, a_i)$. The run is said to be *accepting* if $q_n \in F$. If $u = a_1 a_2 \dots$ is an infinite word of Σ^ω, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 q_2 \dots$ such that for all $i > 0$, we have $q_i \in \Delta(q_{i-1}, a_i)$. A run is said to be *Büchi accepting* if it contains infinitely many accepting states, and *coBüchi accepting* if it contains finitely many non-accepting states. Automata on infinite words will be called Büchi and coBüchi automata, to specify their acceptance condition.

We will note NFA (resp. DFA) for a non-deterministic (resp. deterministic) automaton on finite words, and NCW (resp. DCW) for a non-deterministic (resp. deterministic) coBüchi automaton. An automaton $\mathcal{A}$ is a *safety* automaton if $F = Q$, and every run is accepting.

Non-deterministic automata can be generalized to *alternating* automata, where the transition function associates to each pair $(p, a) \in Q \times \Sigma$ a positive boolean combination of states instead of a disjunction. We refer the reader to [9] for formal definitions and basic constructions on alternating automata.

We also mention the *Rabin condition* on infinite words: it consists of a list of pairs $(E_i, F_i) \in 2^Q \times 2^Q$ and an infinite run is accepting if there is i such that some states from E_i are seen infinitely often and all states from F_i are seen finitely often. Its dual, the negation of a Rabin condition, is called a *Streett condition*. They both generalize the parity condition.

The language of an automaton $\mathcal{A}$, noted $L(\mathcal{A})$, is the set of words on which the automaton $\mathcal{A}$ has an accepting run. Two automata are *equivalent* if they recognise the same language. For a property P of automata (e.g. safety, or coBüchi), a language is said to be P if it is the language of a P automaton.

An automaton $\mathcal{A}$ is *determinisable by pruning* (DBP) if an equivalent deterministic automaton can be obtained from $\mathcal{A}$ by removing some transitions.

An automaton $\mathcal{A}$ is *Good-For-Games* (GFG) if there exists a function $\sigma\colon \Sigma^* \to Q$ (called *GFG strategy*) that resolves the non-determinism of $\mathcal{A}$ depending only on the prefix of the input word read so far: over every word $u = a_1 a_2 a_3 \dots$ (finite or infinite depending on the type of automaton considered), the sequence of states $\sigma(\varepsilon)\sigma(a_1)\sigma(a_1 a_2)\sigma(a_1 a_2 a_3) \dots$ is a run of $\mathcal{A}$ on u, and it is accepting whenever $u \in L(\mathcal{A})$. For instance every DBP automaton is GFG. See [2] for more introductory material and examples on GFG automata.

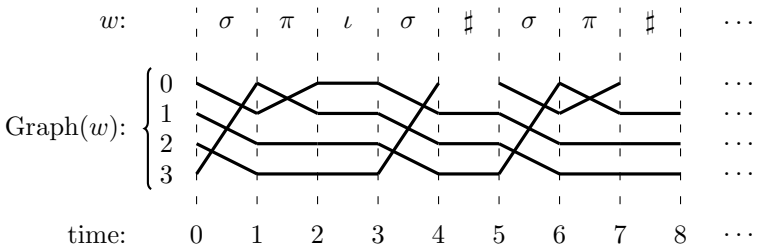
Lemma 1. *GFG automata are closed under the standard union and intersection constructions using cartesian products.*

3 The original family L_n

We start by recalling the family of languages L_n from [15], witnessing an exponential blow-up in the state space for determinisation of GFG automata.

The language L_n is defined on alphabet $\Sigma = \{\iota, \sigma, \pi, \sharp\}$. Each letter represents a permutation of the set $[0, 2n-1]$: ι is the identity, σ is the cycle $(0\ 1\ 2\ \ldots 2n-1)$, π is the transposition $(0\ 1)$, and $\sharp$ is the identity on $[1, 2n-1]$ and is undefined on 0.

An infinite word $w \in \Sigma^\omega$ describes an infinite graph noted Graph(w) with vertices $[0, 2n-1] \times \mathbb{N}$, where letter $w(i)$ representing a permutation α induces edges from vertice (k, i) to $(\alpha(k), i+1)$ for each $k \in [0, 2n-1]$ where α is defined. An example is given below for $n = 2$.



The language L_n is then defined as

$$L_n = \{w \in \Sigma^\omega \mid \text{Graph}(w) \text{ contains an infinite path}\}$$

The infinite path required in the definition of L_n needs not start at time 0. Notice that L_n is suffix-closed and prefix-independent.

Theorem 2. *[15] There is a GFG-NCW with $2n+1$ states for L_n, but any deterministic Streett[3] automaton for L_n has at least $\frac{2^n}{2n+1}$ states.*

However, this example does not settle the blowup problem for languages represented by LTL formulas. Indeed, for any $n \geq 1$, the language L_n is not LTL-definable:

Lemma 3. *For all $n \geq 1$, there is no LTL formula for the language L_n.*

Proof. We use the characterization of LTL-definable languages as aperiodic languages [7]. Let M be the syntactic monoid of L_n, and $h : \Sigma^* \to M$ be the corresponding syntactic morphism. Assume there is $u, v \in \Sigma^*$ such that for infinitely many $k \in \mathbb{N}$, $(u^k v)^\omega \in L_n$ and $(u^{k+1} v)^\omega \notin L_n$. Then we have $h(u)^k \neq h(u)^{k+1}$ for infinitely many k, so the syntactic monoid of L_n cannot be aperiodic, and therefore L_n is not LTL-definable. Hence it suffices to find such u, v to prove that L_n is not LTL-definable. We can take here $u = \sigma$ and $v = \sharp$. Indeed, if k is a multiple of $2n$, we have $(\sigma^k \sharp)^\omega \in L_n$ and $(\sigma^{k+1} \sharp)^\omega \notin L_n$.

[3] Note that the Streett acceptance condition is not specified in [15], but it is in the relevant result of [2]. The Streett condition for $\mathcal{D}$ is needed so that the condition of the form "$\mathcal{A}$ accepts or $\mathcal{D}$ rejects" is Rabin, and the game admits positional strategies.

Moreover, it is shown in [15] that in some sense, the languages L_n essentially constitute the canonical example for coBüchi GFG automata. Indeed, in order to show that deciding whether a NCW is GFG can be done in polynomial time, it is shown that GFG-NCW are very close to the following structure: the automaton deterministically follows a safe path, and when a coBüchi state is encountered, the automaton jumps to another such safe path ; in particular, the safe paths can in some sense be "permuted", and a bad choice of path for the automaton can eventually be corrected by jumping to the right path later, provided the GFG strategy has enough memory to remember how paths were permuted. It is thus plausible that the reason exponential memory is needed in GFG-NCW is the presence of arbitrary permutations, and that this could not happen for LTL-definable languages, where permutations of states are forbidden in the run-DAG of the corresponding automaton [7].

We show in the next section that this is not the case: this exponential blowup result still holds for LTL-definable languages. This gives hope for the use of GFG automata in the context of LTL synthesis.

4 A family of LTL-definable languages K_n with succinct GFG representations

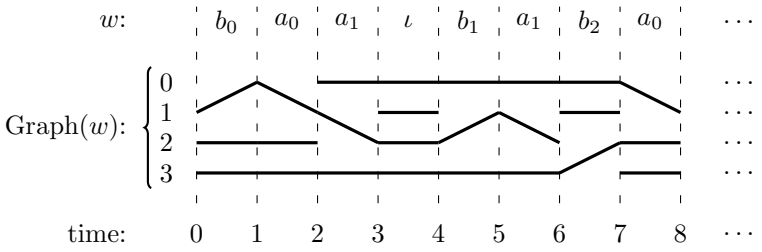
We will define for every $n \geq 1$ a LTL-language K_n. We show that for all $n \geq 1$ there is a GFG-NCW of size $2n+1$ recognizing K_n, but there is no deterministic Streett automaton recognizing K_n of size less than $\frac{2^n}{2n+1}$.

4.1 Definition of the language K_n

The language K_n is defined on alphabet $\Sigma = \{\iota, a_0, a_1, \dots a_{2n-2}, b_0, b_1 \dots b_{2n-2}\}$.

As in L_n, each letter $x \in L$ is mapped to a bipartite graph $G(x)$ describing a partial function $[0, 2n-1] \to [0, 2n-1]$. A word $w = x_1 x_2 \dots$ is in K_n if and only if the DAG Graph(w) with vertices $[0, 2n-1] \times \mathbb{N}$ obtained by concatenating all the slices $G(x_1)G(x_2)\dots$ contains an infinite path, not necessarily starting in $[0, 2n-1] \times \{0\}$, so we define $K_n = \{w \in \Sigma^\omega \mid \text{Graph}(w) \text{ contains an infinite path}\}$.

The graph $G(\iota)$ will represent the identity function. For all $i \in [0, 2n-2]$, the graph $G(a_i)$ maps i to $i+1$, is undefined on $i+1$, and leaves $[0, 2n-1] \setminus \{i, i+1\}$ unchanged. For all $i \in [0, 2n-2]$, the graph $G(b_i)$ maps $i+1$ to i, is undefined on i, and leaves $[0, 2n-1] \setminus \{i, i+1\}$ unchanged. An example of Graph(w) for $n = 2$ is given below.



4.2 Aperiodicity of K_n and succinctness of GFG automaton

Theorem 4. *There is a GFG coBüchi automaton of size $2n+1$ for K_n, but any deterministic Streett automaton for K_n has at least $\frac{2^n}{2n+1}$ states.*

Proof. (Scheme) The same proof scheme as the one from [15] showing the exponential blowup for the family L_n can be used here. We need to adapt it to account for the specificities of K_n, namely modify the construction to avoid crossings of paths.

Lemma 5. *K_n is LTL-definable, via a formula of DAG-size at least exponential in n.*

Proof. (Scheme) LTL definability is shown via the aperiodicity of K_n [7]. We also provide an explicit formula, defined by induction on n, where each induction steps doubles the depth of the formula. Details can be found in Appendix B

Conjecture 6. There is no LTL formula for K_n with DAG-size polynomial in n.

In the next section, we define a specification logic more suited to this setting.

5 A modal logic for eventually safe properties

5.1 The safety logic $S\nu$TL

We recall here the formalism $S\nu$TL, a fragment of linear μ-calculus designed to express safety properties. This fragment has been studied in several works, usually in the branching time setting ; see for example [20], from which we extract (in Theorem 7 below) part of the characterization of $S\nu$TL as the safety fragment of the μ-calculus. A similar characterization for LTL is sketched in the conclusions of [19].

Formulas of $S\nu$TL are given by the following syntax, where a stands for letters of Σ.

$$\varphi := a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \odot\varphi \mid X \mid \nu X.\varphi$$

If φ is a formula of $S\nu$TL, we will note $[\![\varphi]\!]$ its semantic as a linear μ-calculus formula.

Theorem 7. *A language is definable in $S\nu$TL if and only if it is a safety language.*

Proof. (Scheme) For the left to right implication, we follow a more general construction that has been proposed in [20], for branching μ-calculus. In our case, the states of the constructed alternating safety automaton are subformulas of the input $S\nu$TL formula ψ, and the transition function describes the subformulas that should be true after reading a letter, using alternation to encode the disjunctions and conjunctions. Since alternating safety automata are equivalent to safety languages through standard powerset constructions we are done. For the right to left implication, we build a $S\nu$TL formula from a non-deterministic safety automaton by encoding into the formula the transition function; loops in the automaton are encoded using the operator νX.

5.2 The logic $E\nu$TL for eventually safe properties

We introduce here a second semantic $[\![\cdot]\!]_E$ for $S\nu$TL formulas, called the *eventual semantic*, in the following way:

$$[\![\varphi]\!]_E := \{uw \in \Sigma^\omega \mid w \in \mathit{suff}([\![\varphi]\!])\}.$$

That is to say, $[\![\varphi]\!]_E$ denotes infinite words that have a suffix that is also a suffix of $[\![\varphi]\!]$. This logic can be seen as a way to specify what we mean by a safe behaviour, letting the semantic automatically generate the language of eventually safe behaviours. Notice that since GFG automata are closed under union and intersection by Lemma 1, the technique introduced in this paper can be combined with others in order to treat more advanced specifications.

We call $E\nu$TL the logic $S\nu$TL equipped with the eventual semantic.

Lemma 8. *A language is definable in $E\nu$TL if and only if it is of the form $\Sigma^* L_{safe}$, where L_{safe} is a suffix-closed safety language.*

Proof. If L is defined in $E\nu$TL via the formula φ, then $L = \Sigma^* \mathit{suff}([\![\varphi]\!])$, which is of the wanted form. Conversely, if $L = \Sigma^* L_{safe}$ with L_{safe} a suffix-closed safety language, then by Theorem 7 there is a formula φ of $S\nu$TL such that $[\![\varphi]\!] = L_{safe}$, and we obtain $L = [\![\varphi]\!]_E$.

Let us call *ESafe* the class of such languages. Notice that $E\nu$TL uses a syntax for safety properties, but with our semantics, the languages defined by $E\nu$TL are actually liveness properties [1]: if $L \in \mathit{ESafe}$, any finite word can be extended to a word in L.

5.3 Properties of the class *ESafe*

The following theorem states that the class *ESafe* captures exactly prefix-independent coBüchi languages, and that it is equivalent to represent a language from *ESafe* directly with a NCW, or via its suffix-closed safety language L_{safe}.

Theorem 9. *ESafe is equal to the class of prefix-independent coBüchi languages. Moreover, if $L = \Sigma^* L_{safe}$ is accepted by a NCW $\mathcal{C}$, we can build a non-deterministic safety automaton $\mathcal{A}_{safe}$ from $\mathcal{C}$ recognizing L_{safe} with the same number of states. Conversely, if we have a non-deterministic safety automaton for L_{safe}, we can build a NCW $\mathcal{C}$ for L with one more state.*

Lemma 10. *Given a regular language L, it is decidable whether it is in ESafe. If L is given by a DCW, the problem is in NL, whereas it is PSPACE-complete if L is given by a NCW.*

The complexity of deciding whether an arbitrary regular language (given by various models of automata and LTL formulas) is coBüchi-recognizable, and obtaining an equivalent DCW or NCW automaton, is studied in [3]. This completes the picture for the problem of deciding whether an arbitrary language can be represented using $E\nu$TL.

5.4 A succinct $E\nu$TL formula for the language K_n

The formula we aim to build will describe the safety languages of words for which the path starting in $(0,0)$ is infinite. It recognizes K_n via the eventually safe semantic. We use the weak until operator **WU** as syntactic sugar, defined as $\varphi \mathbf{WU} \psi := \nu X.\psi \vee (\varphi \wedge \odot X)$.

The pure LTL formula for K_n given in Section B.2 has a DAG-size at least exponential in n. The formulas we will define here will instead be linear in n. Let $N = 2n-1$. We reuse the subalphabets $(\alpha_i)_{0 \leq i \leq N}$ defined in Section B.2. We define inductively the formulas ψ_i for i from N to 0, each one containing X_{i-1} as a free variable, except for ψ_0 which is closed:

$$\begin{aligned} \psi_N &= \nu X_N.((\alpha_N \wedge \odot X_N) \vee (b_{N-1} \wedge \odot X_{N-1})) \\ \text{For } 0 < i < N : \psi_i &= \nu X_i.((\alpha_i \wedge \odot X_i) \vee (a_i \wedge \odot \psi_{i+1}) \vee (b_{i-1} \wedge \odot X_{i-1})) \\ \psi_0 &= \nu X_0.((\alpha_0 \wedge \odot X_0) \vee (a_0 \wedge \odot \psi_1)) \end{aligned}$$

We finally define $\Phi := \psi_0$.

Lemma 11. *The formula Φ has size linear in n, and $[\![\Phi]\!]_E = K_n$.*

We note that a similar formula can be explicited for the original family of languages (L_n) witnessing exponential succinctness of GFG automata. This formula allows to encode the examples of specifications on interacting agents given in the introduction. Details can be found in Appendix D.2

5.5 From $E\nu$TL to GFG coBüchi automata

In this section, we describe a general algorithm for translating any $E\nu$TL formula to an equivalent GFG-NCW. Recall that since GFG automata are sound for synthesis [11], this translation can be used as a blackbox for solving synthesis of *ESafe* properties, specified via the logic $E\nu$TL. As explained before, this translation can also be used in a modular way, and combined with other deterministic or GFG automata as shown in Lemma 1. We now describe the algorithm, taking as input an $E\nu$TL formula ψ. We can view ψ as a $S\nu$TL formula and build an alternating safety automaton $\mathcal{A}_{\text{alt}}$ recognizing $[\![\psi]\!]$ as described in the proof of Theorem 7.

We use a powerset construction to obtain an equivalent non-deterministic safety automaton $\mathcal{A}_{\text{nd}}$. Determinizing $\mathcal{A}_{\text{nd}}$ to a deterministic safety automaton A_{det} through another powerset construction is standard. Since A_{det} can be equivalently seen as a safety DFA, it can be minimized into a safety deterministic automaton $A_{\min} = (Q, \Sigma, q_0, \Delta)$ using standard techniques. Minimization techniques can also be applied on the intermediate automaton $\mathcal{A}_{\text{nd}}$, for instance using bisimulation equivalence [18]. Here we omitted the accepting states of $A_{\min}$, since all runs are accepting. We assume that all states of $A_{\min}$ are reachable from q_0.

We will now build a GFG coBüchi automaton $\mathcal{C} = (Q', \Sigma, q_0, \Delta, F)$ for $[\![\psi]\!]_E$, based on $A_{\min}$. We take $Q' = Q \cup \{\bot\}$, $F = Q$, and

$$\Delta' = \Delta \cup \{(p, a, \bot) \mid \forall q \in Q, (p, a, q) \notin \Delta\} \cup \big(\{\bot\} \times \Sigma \times Q\big)$$

The following theorem states that the algorithm is correct.

Theorem 12. *$\mathcal{C}$ is a GFG-NCW for $[\![\psi]\!]_E$.*

Proof. (Scheme) $\mathcal{C}$ will deterministically follow paths made of safe transitions, and will go to $\bot$ when the path it is currently following is cut. The only non-determinism to resolve is: where to jump from $\bot$? It suffices to jump to the path that has been uncut for the longest time. Details can be found in Appendix D.3.

Remark 13. An alternative construction where strongly connected components of $\mathcal{A}_{\text{nd}}$ are determinized separately is also possible and allows more optimizations, we discuss this in Appendix D.4. This can yield smaller GFG automata in cases where the size of the biggest strongly connected component of $\mathcal{A}_{\text{nd}}$ is small in front of its total size, or if some components cover the safe languages of others.

The complexity of this algorithm is doubly exponential in terms of number of states. The following theorem shows that this cannot be avoided.

Theorem 14. *The translation of $E\nu$TL formulas to GFG-NCW is doubly exponential.*

Proof. This result for general LTL formulas has been proven in [4] using a language family $(\mathcal{L}_n)$ defined in [16], itself adapted from a language of finite words given in [6]. However, the particular structure of the class *ESafe* prevents us from using the results of [4, 16, 6] as blackboxes. The language $\mathcal{L}_n$ as defined by the LTL formula from [16] is $F_n\sharp^\omega$ where F_n is the language defined by $F_n = \{\{0,1,\#\}^* \cdot \# \cdot w \cdot \# \cdot \{0,1,\#\}^* \cdot \$ \cdot w \mid w \in \{0,1\}^n\}$.We will use similar ideas here, while taking care of the special semantic of $E\nu$TL.

Let $\Sigma = \{0,1,\#,\$,\flat\}$. We change the formula from [16] so that we iterate the closure of F_n, with $\flat$ as separator. Intuitively, the $S\nu$TL formula φ_n states that any word on $\{0,1\}^*$ immediately following a \$ has length n, is followed by a $\flat$, and is identical to a word that occurred between the last $\flat$ and the \$ following it. Let $\Theta = 0 \vee 1$ and $\Theta_\sharp = \Theta \vee \sharp$, we define the following formulas:

$$
\begin{aligned}
&\varphi_{\text{word}}(X) = \Theta \wedge \odot(\Theta \wedge \overset{n}{\cdots} \odot(\Theta \wedge \odot(\flat \wedge \odot X)) \cdots) \\
&\varphi_{\text{i,x}} = (\odot^i x \wedge (\Theta_\sharp \mathbf{WU}(\$ \wedge \odot^i x))) \qquad \text{for } 1 \leq i \leq n \text{ and } x \in \{0,1\} \\
&\varphi_{\text{match}} = \textstyle\bigwedge_{1 \leq i \leq n} (\varphi_{\text{i},0} \vee \varphi_{\text{i},1}) \\
&\varphi_n = \nu X. \Theta_\sharp \mathbf{WU}[\# \wedge \varphi_{\text{match}} \wedge (\Theta_\sharp \mathbf{WU}(\$ \wedge \odot \varphi_{\text{word}}(X)))]
\end{aligned}
$$

The formula $\varphi_{\text{word}}(X)$ checks for $\{0,1\}^n \flat X$, where X is a free variable. The formula $\varphi_{\text{i,x}}$ checks that i^{th} letter is x and that it is again the case after the next \$. The formula φ_{match} enforces that the current word $u \in \{0,1\}^n$ must be matched by a $\$u$ at the next \$. Finally, φ_n ensures that before the next \$, we encounter some $\sharp u$ where u is further matched by $\$u\flat$, after which we reiterate the constraint. Notice that $|\varphi_n|$ is quadratic in n, and $[\![\varphi_n]\!]_E \cap (\Sigma^*\flat)^\omega = \Sigma^*(F_n\flat)^\omega$. Let us assume that we have a GFG-NCW automaton $\mathcal{C}$ for $[\![\varphi_n]\!]_E$. We show that $\mathcal{C}$ must have at least 2^{2^n-1} states. Let Q be the set of states of $\mathcal{C}$ and $\sigma : \Sigma^* \to Q$ be the GFG strategy of $\mathcal{C}$. Let us assume that $|Q| < 2^{2^n-1}$. We call the *type* of a finite run of $\mathcal{C}$ the pair (c, q), where c is a bit specifying whether a rejecting state

has been seen, and q is the last state of the run. The number of possible types is $2|Q| < 2^{2^n}$. To any set of words $X = \{u_1, u_2, \ldots, u_k\} \subseteq \{0,1\}^n$, we associate a word $w_X = \#u_1\#u_2\#\ldots\#u_k$, where the u_i's are lexicographically sorted. Since there are 2^{2^n} such sets, there must be $X_1 \neq Y_1 \subseteq \{0,1\}^n$ such that $\sigma(w_{X_1})$ and $\sigma(w_{Y_1})$ have same type. Without loss of generality let $w_1 \in X_1 \setminus Y_1$, we have $w_{X_1}\$w_1 \in F_n$ but $w_{Y_1}\$w_1 \notin F_n$. Again, there are X_2, Y_2, q_2, w_2 such that $\sigma(w_{X_1}\$w_1\flat w_{X_2})$ and $\sigma(w_{Y_1}\$w_1\flat w_{Y_2})$ have same type on the suffix starting with $\$w_1\flat$, and $w_2 \in X_2 \setminus Y_2$. By iterating this construction, we build two infinite words $v = w_{X_1}\$w_1\flat w_{X_2}\$w_2\flat\ldots$ and $v' = w_{Y_1}\$w_1\flat w_{Y_2}\$w_2\flat\ldots$ such that σ accepts v if and only if σ accepts v', but we have $v \in [\![\varphi_n]\!]_E$ while $v' \notin [\![\varphi_n]\!]_E$. We obtain a contradiction with the fact that $\mathcal{C}$ recognizes $[\![\varphi_n]\!]_E$ with GFG strategy σ.

However, our algorithm can perform well in practice, as witnessed by the languages K_n and L_n. Indeed, if the input is the formula Φ from section 5.4 (resp. φ_0 from appendix D.2) describing the language K_n (resp. L_n), the algorithm computes an automaton of size linear in n, while any deterministic automaton would be exponential, by Theorem 4 (resp. Theorem 2).

Conclusion

We showed that GFG automata still enjoy an exponential succinctness gap compared to deterministic automata for the class of LTL-definable languages, by giving a family of languages (K_n) witnessing this gap. However, the LTL formula we give for K_n is exponential in n, while there is an equivalent μ-calculus formula that is linear in n. We conjecture that the family (K_n) can be used as a witness to prove an exponential gap in succinctness between the linear μ-calculus and LTL. We defined and studied a class *ESafe* of eventually safe languages, a natural compromise between safety and liveness specifications. We defined a fragment of linear μ-calculus, the logic $E\nu$TL, that describes the class *ESafe*, and can be translated to GFG coBüchi automata.

The idea of using automata that are allowed to non-deterministically jump to a new path to improve performances in LTL synthesis was implemented in [10], but the so-called "shift automaton" was based on a powerset construction, and the notion of Good-for-Games was not identified, replaced by the weaker condition of being Determinisable by Pruning. Prior to the discovery of succinctness of GFG automata, a GFG-based algorithm for model-checking of Markov decision process has been implemented in [12], where it turned out that this particular approach was not more efficient than standard ones.

As future work, we plan to implement our approach that makes use of newly discovered features of GFG automata, in order to compare benchmarks with those present in [10, 12]. A related open challenge is to find fragments of LTL or other logics that can be translated to GFG automata with only a single exponential blowup.

References

1. Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181 – 185, 1985.
2. Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 89–100, 2013.
3. Udi Boker and Orna Kupferman. Co-Büching Them All. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 184–198, 2011.
4. Udi Boker, Orna Kupferman, and Michał Skrzypczak. How deterministic are Good-For-Games automata? In *FSTTCS 2017 - 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, Leibniz International Proceedings in Informatics (LIPIcs), 2017.
5. Julius Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
6. Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
7. Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives, Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.
8. Javier Esparza, Jan Kretínský, and Salomon Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 384–393, 2018.
9. Abdelaziz Fellah, Helmut Jürgensen, and Sheng Yu. Constructions for alternating finite automata. *International journal of computer mathematics*, 35(1-4):117–132, 1990.
10. Aidan Harding, Mark Ryan, and Pierre-Yves Schobbens. A new algorithm for strategy synthesis in LTL games. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 477–492. Springer, 2005.
11. Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 395–410, 2006.
12. Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. Are good-for-games automata good for probabilistic model checking? In *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, pages 453–465, 2014.
13. Jan Kretínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 567–577, 2018.
14. Denis Kuperberg and Anirban Majumdar. Width of non-deterministic automata. In *35th International Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 29th - March 3rd, 2018, Caen, France*, 2018.

15. Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 299–310, 2015.
16. Orna Kupferman and Moshe Y. Vardi. From linear time to branching time. *ACM Trans. Comput. Log.*, 6(2):273–294, 2005.
17. Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.
18. Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.
19. A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.
20. Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 8(2):359, 2001.
21. Shufang Zhu, Lucas M Tabajara, Jianwen Li, Geguang Pu, and Moshe Y Vardi. A Symbolic Approach to Safety LTL Synthesis. In *Haifa Verification Conference*, pages 147–162. Springer, 2017.

A GFG automata

A.1 Proof of Lemma 1

Let $\mathcal{A}_1 = (Q_1, \Sigma, q_1, \Delta_1, \alpha_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, q_2, \Delta_2, \alpha_2)$ be two GFG automata. Here α_1 and α_2 are arbitrary acceptance conditions specifying the set of accepting runs: $\alpha_i \subseteq Q_i^\omega$ for $i \in \{1, 2\}$. Let $\sigma_i : \Sigma^* \to Q_i$ be a GFG strategy for $\mathcal{A}_i$.

We build a GFG automaton $\mathcal{A} = (Q, \Sigma, \Delta, q_0, \alpha)$ for $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ using the usual product construction on automata:

- $Q = Q_1 \times Q_2$ and $q_0 = (q_1, q_2)$
- $\Delta = \{(p_1, p_2) \xrightarrow{a} (p_1', p_2') \mid p_1 \xrightarrow{a} p_1' \in \Delta_1 \text{ and } p_2 \xrightarrow{a} p_2' \in \Delta_2\}$
- $\alpha = \{(\rho_1, \rho_2) \mid \rho_1 \in \alpha_1 \text{ or } \rho_2 \in \alpha_2\}$

It remains to show that $\mathcal{A}$ is GFG. We define $\sigma : \Sigma^* \to Q$ for $\mathcal{A}$ by $\sigma(u) = (\sigma_1(u), \sigma_2(u))$. It is straightforward to verify that σ is a GFG strategy for $\mathcal{A}$: whenever $w \in L(\mathcal{A})$, we have that w can be accepted by $\mathcal{A}_1$ or $\mathcal{A}_2$, so either $\sigma_1(w)$ or $\sigma_2(w)$ produces an accepting run, hence $\sigma(w)$ is accepting as well.

Notice that the accepting condition α of $\mathcal{A}$ is a union of α_1 and α_2. This means that depending on the nature of α_1 and α_2, the condition α may be of the same complexity or get more complicated. For instance if α_1 and α_2 are Büchi conditions, then α is Büchi as well. If α_1 and α_2 are parity conditions, α becomes a Rabin condition, and may be transformed into a parity condition with more ranks than α_1 and α_2.

Notice that if α_1 and α_2 are coBüchi, an alternative construction yields a GFG coBüchi automaton, using the standard construction for the union of coBüchi automata (a coBüchi state of the product is witnessed when both components have seen a coBüchi state). The resulting number of states is still in $O(|Q_1| \cdot |Q_2|)$ in this case.

A similar construction shows that GFG automata are closed under intersection. In the same way, Büchi (resp. coBüchi) GFG automata can be intersected while keeping the acceptance condition, and parity GFG automata can be intersected at the price of increasing the number of parity ranks.

A.2 Proof of Theorem 4

We show that there is a GFG-NCW for K_n with $2n + 1$ states, and that any deterministic Streett automaton for K_n has size at least $\frac{2^n}{2n+1}$.

This proof closely follows the proof used in [15] for the languages L_n, adapted here for the languages K_n. The main changes are gathered at the end of the proof, such as Lemma 20.

We start by showing that there is a linear-size GFG-NCW $\mathcal{C}_n$ for the language K_n.

The set of states of the automaton $\mathcal{C}_n$ is $Q = \{\bot, 0, 1, 2, \ldots, 2n-1\}$. The states $[0, 2n-1]$ are deterministic: reading $a \in \Sigma$ in such a state q the automaton

moves to the successive state according to the function represented by a (or to $\bot$ if this function is undefined on q). The state $\bot$ is non-deterministic — the automaton can move from $\bot$ over any letter $a \in \Sigma$ to any state $q' \in [0, 2n-1]$. Let the initial state of $\mathcal{C}_n$ be $\bot$ and the rejecting transitions be those of the form $\bot \xrightarrow{a} q'$.

Note that every accepting run of $\mathcal{C}_n$ over an ω-word α indicates an infinite path in Graph(α). Therefore, we obtain the following fact.

Fact 15 $L(\mathcal{C}_n) \subseteq K_n$.

Lemma 16. *$\mathcal{C}_n$ is a GFG automaton recognizing the language K_n.*

Proof. It is enough to construct a function $\sigma\colon \Sigma^* \to Q$ that for every ω-word $\alpha \in K_n$ produces an accepting run of $\mathcal{C}_n$ over α — it will prove that $K_n \subseteq L(\mathcal{C}_n)$ and that $\mathcal{C}_n$ is GFG. The strategy σ will use a finite memory (of size 2^{2n}) under the form of a nonempty subset of $Q \setminus \{\bot\}$ that will be updated at every transition. This means we want to define a function $M\colon \Sigma^* \to 2^{Q\setminus\{\bot\}}$. We will define σ and M inductively with $\sigma(\epsilon) = \bot$ and $M(\epsilon) = Q \setminus \{\bot\}$.

Let us define the memory function M first. We define

$$M(wa) = \{\delta(q, a) | q \in M(w) \text{ and } \delta(q, a) \neq \bot\}$$

if this set is not empty, and $M(wa) = Q \setminus \{\bot\}$ otherwise. Intuitively, $M(w)$ tracks all the paths in Graph(w) that have not been cut so far, and when all such paths have been cut, it reinitializes. This information will be used by the strategy σ.

Let σ follow deterministically the transitions of $\mathcal{C}_n$ for all the states $q \neq \bot$: if $\sigma(w) = q$, then $\sigma(wa) = \delta(q, a)$ for any letter a. It remains to define $\sigma(wa)$ if $\sigma(w) = \bot$ and a successive letter a is given. In this case, we consider the set $M(wa)$, and $\sigma(wa)$ is defined to be the smallest element of $M(wa)$. Note that this definition ensures that $\sigma(w) \in M(w)$ for every w such that $\sigma(w) \neq \bot$.

Assume that $\alpha \in K_n$. We need to prove that σ produces an accepting run of $\mathcal{C}_n$ over α. Let $p_1, p_2, \ldots, p_m$ be the set of infinite paths in Graph(α) (we know that $1 \leq m \leq n$). By definition of M, the sets $M(\alpha_{<k})$ for $k \in \mathbb{N}$ will eventually contain only states belonging to some of the infinite paths p_i. Let k_0 such that for all $k \geq k_0$, for all $p \in M(\alpha_{<k})$, the node (p, k) is part of an infinite path in Graph(α). Assume that $\sigma(\alpha_{<k_0}) \neq \bot$ (if this is not the case, we replace k_0 by $k_0 + 1$). Since $\sigma(\alpha_{<k_0})$ belongs to $M(\alpha_{<k_0})$, then it will follow in $\mathcal{C}_n$ the corresponding infinite path and α will be accepted.

We assume for the sake of contradiction that there exists a deterministic Streett automaton $\mathcal{D}$ recognising K_n that has strictly less than $\frac{2^n}{2n+1}$ states. By Theorem 4 from [2], it means that we can use $\mathcal{D}$ as a memory structure for the automaton $\mathcal{C}_n$ to recognise K_n. Therefore, we focus on the product $\mathcal{C}_n \times \mathcal{D}$ with the acceptance condition taken from $\mathcal{C}_n$. What is important is that $\mathcal{C}_n \times \mathcal{D}$ has to follow the transitions of $\mathcal{C}_n$. We know that $\mathcal{C}_n \times \mathcal{D}$ is a deterministic coBüchi automaton with strictly less than 2^n states and $L(\mathcal{C}_n \times \mathcal{D}) = K_n$.

We will use the symbol ρ to denote finite and infinite runs of $\mathcal{C}_n \times \mathcal{D}$. For a given run ρ there are possibly many ω-words α that induce this run, since only the sequence of states is considered in ρ.

The rest of the argument aims at providing an ω-word α that belongs to K_n but is rejected by the product automaton $\mathcal{C}_n \times \mathcal{D}$. Intuitively, the construction of α requires to balance between two aims: we need to infinitely often force the product automaton $\mathcal{C}_n \times \mathcal{D}$ to take a rejecting transition of $\mathcal{C}_n$ but at the same time to ensure that there is at least one infinite path in $\mathrm{Graph}(\alpha)$. The ω-word α, an infinite path in $\mathrm{Graph}(\alpha)$, and the rejecting run of $\mathcal{C}_n \times \mathcal{D}$ over α will be constructed as a limit of inductively constructed finite approximations. We will not control exactly the way $\mathcal{C}_n \times \mathcal{D}$ works in every position of our approximation, we will be interested only in some *checkpoints* controlled by partial runs.

Definition 17. *A* partial run *is a finite partial mapping* $\tau\colon \omega \rightharpoonup Q^{\mathcal{C}_n} \times Q^{\mathcal{D}}$ *such that* $\tau(0)$ *is defined and equal to* $(\bot, q_{\mathbf{I}}^{\mathcal{D}})$.

A partial run τ *is* rejecting *if all its states are of the form* $(\bot, m)$.

We denote by $\tau \subseteq \rho$ *the fact that a run* ρ *agrees with* τ *wherever* τ *is defined.*

The length *of* τ *is the maximal moment of time* k *such that* $\tau(k)$ *is defined.*

Note that the domain of a partial run τ does not have to be an initial segment of ω. The following definition is crucial.

Definition 18. *Let* τ *be a partial run of length* k. *We say that a value* $i \in [0, 2n-1]$ *is* alive in τ *if there exists an* ω*-word* α *such that for the run* ρ *of* $\mathcal{C}_n \times \mathcal{D}$ *over* α *we have* $\tau \subseteq \rho$ *and there exists a path* $p\colon [0,k] \to [0, 2n-1]$ *in* $\mathrm{Graph}(\alpha)$ *that starts in the moment of time* 0 *and ends in the moment of time* k *with the value* i *(i.e.* $p(k) = i$*).*

Note that in the above definition we actually care only about the first k letters of α. However, it is cleaner to consider ω-words α here.

Figure 1 depicts a partial run and a witness that the value $i = 2$ is alive.

Our aim is to construct a sequence of partial rejecting runs of increasing lengths $\tau_0 \subset \tau_1 \subset \ldots$ such that for all $\ell \in \mathbb{N}$ there are at least n alive values in τ_ℓ. This will give a contradiction with our assumptions by the following Lemma.

Lemma 19. *Assume that there exists a sequence of partial rejecting runs* $\tau_0 \subset \tau_1 \subset \ldots$ *of increasing lengths such that for all* $\ell \in \mathbb{N}$ *there exists an alive value in* τ_ℓ. *Then there exists an* ω*-word* $\alpha \in K_n$ *such that the run* ρ *of* $\mathcal{C}_n \times \mathcal{D}$ *over* α *is rejecting.*

Proof. Let k_ℓ be the length of τ_ℓ. Take any ℓ and assume that i_ℓ is a value that is alive in τ_ℓ. Observe that it is witnessed by:

- an ω-word α_ℓ,
- a run ρ_ℓ of $\mathcal{C}_n \times \mathcal{D}$ over α_ℓ, such that $\tau_\ell \subset \rho_\ell$,
- a path $p_\ell\colon [0, k_\ell] \to [0, 2n-1]$ in $\mathrm{Graph}(\alpha_\ell)$ with $p_\ell(k_\ell) = i_\ell$.

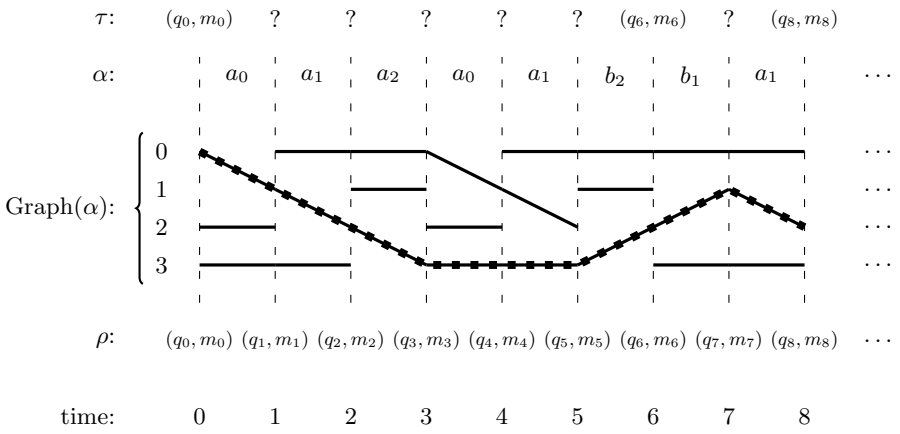


Fig. 1. An example of a partial run τ and an ω-word α that witnesses the fact that 2 is alive in τ. The run ρ is the run of $\mathcal{C}_n \times \mathcal{D}$ over α and the states of ρ and τ agree wherever defined. The dashed path is the path witnessing that 2 is alive in τ.

Now we take a subsequence of $(\alpha_\ell, \rho_\ell, p_\ell)_{\ell \in \mathbb{N}}$ that is point-wise convergent to a triple

$$(\alpha, \rho, p) \in \left(\Sigma \times \left(Q^{\mathcal{C}_n} \times Q^{\mathcal{D}}\right) \times [0, 2n-1]\right)^\omega, \text{ such that :}$$

- ρ is the run of $\mathcal{C}_n \times \mathcal{D}$ over α,
- for infinitely many ℓ we have $\tau_\ell \subseteq \rho$,
- p encodes an infinite path in Graph(α).

To formally construct (α, ρ, p) we can proceed similarly as in the proof of König's lemma. We fix $(\alpha(i), \rho(i), p(i))$ inductively for $i = 0, 1, \ldots$. At each moment we require that infinitely many $(\alpha_\ell, \rho_\ell, p_\ell)$ agree with (α, ρ, p) on the first i positions. Since for each i there are only finitely many choices of $(\alpha(i), \rho(i), p(i))$ so we can fix these values in such a way that still infinitely many $(\alpha_\ell, \rho_\ell, p_\ell)$ agree with them.

By the properties of (α, ρ, p) we know that ρ is rejecting as it contains infinitely many times a state of the form $(\bot, m)$. On the other hand, $\alpha \in K_n$ because p is a witness that Graph(α) contains an infinite path.

What remains is to construct the sequence τ_ℓ inductively. Our inductive assumption is that τ_ℓ is a partial rejecting run and the values $1, 3, 5, \ldots, 2n-1$ are alive in τ_ℓ (note that there are n such values). We put $\tau_0 = \left[0 \mapsto (\bot, q_{\mathbf{I}}^{\mathcal{D}})\right]$. Clearly τ_0 satisfies the inductive assumption (in fact all the values $i = 0, \ldots, 2n-1$ are alive in τ_0).

Let k_ℓ be the length of τ_ℓ. We construct $\tau_{\ell+1}$ from τ_ℓ by applying some words to the last state $(\bot, m_\ell) = \tau_\ell(k_\ell)$ of τ_ℓ and observing the behaviour of $\mathcal{C}_n \times \mathcal{D}$.

The rest of the proof differs from the proof from [15], as we have to account for the specificity of K_n, namely that paths cannot cross.

Observe that there are $N = 2^n$ words $u_1, \ldots, u_N \in \Sigma^*$ that encode total functions $f : \{1, 3, 5 \ldots, 2n-1\} \to [0, 2n-1]$ such that for all $i \in \{1, 3, 5 \ldots, 2n-1\}$, we have $f(i) \in \{i, i-1\}$. Such a word for a function f can be built as $b_{i_1} b_{i_2} \ldots b_{i_k}$ where $\{i_1, \ldots i_k\} = \{i \in \{0, 2, \ldots, 2n-2\} \mid f(i+1) = i\}$.

We can assume that all the words $u_1, \ldots, u_N$ are of equal length by padding them with ι. Since there are strictly less than $N = 2^n$ states of $\mathcal{C}_n \times \mathcal{D}$, there are two distinct such words u, u' leading from $(\bot, m_\ell)$ to the same state (q'_ℓ, m'_ℓ) of $\mathcal{C}_n \times \mathcal{D}$. By the construction of $\mathcal{C}_n \times \mathcal{D}$ we know that $q'_\ell \in \{0, \ldots, 2n-1\}$.

Assume that the functions corresponding to u and u' differ on $2i+1$, i.e. one of them maps $2i+1$ to $2i$ and the other to $2i+1$. Let X be the set of the values $\{u(1), u(3), \ldots, u(2n-3), u(2n-1), u'(2i+1)\}$ (we write here $u(i')$ for the value assigned to i' by the permutation corresponding to u, the same for u'). By the above observations X contains exactly $n+1$ elements.

Lemma 20. *There exists $w \in \Sigma^*$ that:*

- *cuts the path from q'_ℓ at the last letter,*
- *maps $X \setminus \{q'_\ell\}$ to $1, 3, 5, \ldots, 2n-1$ if $q'_\ell \in X$,*
- *maps X to $1, 3, \ldots, 2n-1$, and some even integer $2j$ if $q'_\ell \notin X$.*

Proof. Let $A = \{a_i \mid i \in [0, 2n-2]\}$ and $B = \{b_i \mid i \in [0, 2n-2]\}$.

Notice that if $Y = \{y_1, \ldots, y_k\}$ and $Z = \{z_1, \ldots, z_k\}$ are subsets of $[0, 2n-1]$ with $y_i < y_{i+1}$ and $z_i < z_{i+1}$ for all $i \in [0, 2n-2]$, there exists a word $v \in \Sigma^*$ mapping y_i to z_i for all $i \in [1, k]$. It suffices to take $v = w_1 w_2 \ldots w_k w'_k w'_{k-1} \ldots w'_1$ where $w_i \in B^*$ maps y_i to $i-1$ and uses no unnecessary letters, and $w'_i \in A^*$ maps $i-1$ to z_i in the same way. Proceeding in this order avoids the cutting of any path starting in Y.

Let us assume first that $q'_\ell \in X$. If $q'_\ell \neq \min X$, we can find a word v mapping X to $1, 3, \ldots, 2n-1$, and an even integer $2j$, such that q'_ℓ is mapped to $2j+1$. Taking $w = va_{2j}$ completes the proof in this case. If $q'_\ell = \min X$, we can choose v to map X to $1, 3, \ldots, 2n-1$ and 2, with q'_ℓ mapped to 1, and take $w = vb_1$.

Assume now that $q'_\ell \notin X$. If $q'_\ell > \min X$. We can find a word v mapping X to $1, 3, \ldots, 2n-1$, and two even integers $2j$ and $2j'$, such that q'_ℓ is mapped to $2j'+1$. We then take $w = va_{2j'}$. Finally, if $q'_\ell < \min X$, we choose v to map X to $1, 3, \ldots, 2n-1$ and $2, 4$, with q'_ℓ mapped to 1, and take $w = vb_1$.

Since w cuts the path from q'_ℓ at the last letter, it means that after reading w from the state (q'_ℓ, m'_ℓ) the automaton $\mathcal{C}_n \times \mathcal{D}$ reaches a state of the form $(\bot, m_{\ell+1})$. For an illustration of these functions, see Figure 2.

Fact 21 *Consider $\tau_{\ell+1}$ defined as $\tau_\ell \cup \big[k_\ell + |u| + |w| \mapsto (\bot, m_{\ell+1})\big]$. By the definition $\tau_\ell \subset \tau_{\ell+1}$, $\tau_{\ell+1}$ is rejecting, and all the values $1, 3, \ldots, 2n-1$ are alive in $\tau_{\ell+1}$ (it is witnessed by the fact that these values were alive in τ_ℓ and by the words uw and $u'w$).*

Therefore, we have constructed $\tau_{\ell+1}$ that satisfies the inductive invariant. This concludes the inductive construction of the sequence $(\tau_\ell)_{\ell \in \mathbb{N}}$. By Lemma 19 it finishes the proof of Theorem 4.

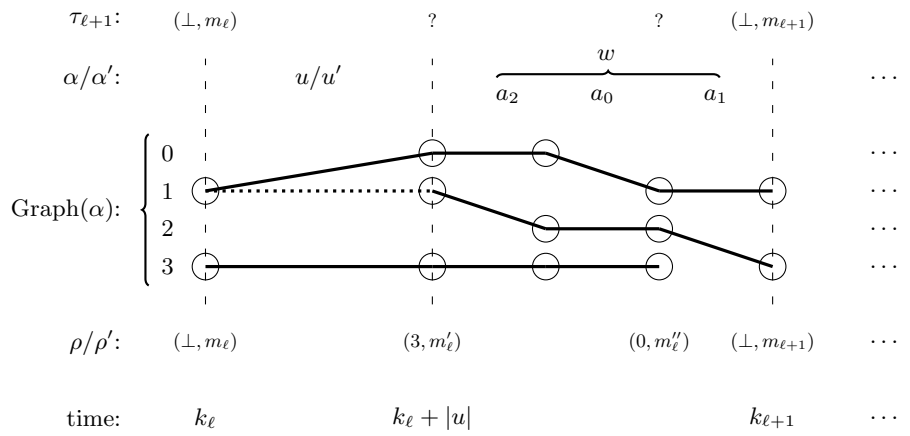


Fig. 2. The behaviour of $\mathcal{C}_n \times \mathcal{D}$ over uw and $u'w$. The alive values are in circles, only edges between the alive values are drawn. The dashed edge corresponds to the action of the word u' on the value 1 (u and u' differ on this value). X is the set of values in circles at the moment of time $k_\ell + |u|$. $q'_\ell = 3$ is cut by the last letter of $w = a_1a_0a_2$, the other elements of X are mapped to 1 and 3.

B Describing K_n in LTL

B.1 LTL-definability of K_n

We first show that the language K_n is LTL-definable using a common characterization of LTL-definable languages [7]. We will give an explicit formula for K_n in the next section.

Lemma 22. *For every word $w \in \Sigma^*$, there exists $X_w \subseteq [0, 2n-1]$ such that for all $N \geq 2n$, $x \in X_w$ and $y \notin X_w$,* Graph(w^N) *has a path from $(x, 0)$ to $(x, N|w|)$, and the path starting in $(y, 0)$ is cut.*

Proof. Let $X_w = \{x \in [0, 2n-1] \mid \text{Graph}(w) \text{ has a path from } (x, 0) \text{ to } (x, |w|)\}$. Let $y \notin X_w$, and let us prove that the path starting in $(y, 0)$ is cut in Graph(w^{2n}), which will conclude the proof. If the path in Graph(w) starting in $(y, 0)$ is cut, then we are done. If it is not, then it ends in $y_1 \neq y$. Assume that $y_1 > y$ (the other case being symmetric). We have $y_1 \notin X_w$, for if this was the case, the path ending in y_1 would start in y_1, and this would imply $y_1 = y$. Moreover, the path in Graph(w) starting in y_1 is either cut, in which case we are done since the path in Graph(w^2) starting in $(y, 0)$ would be cut, or ends in $(y_2, |w|)$ with $y_2 > y_1$: indeed, if we had $y_2 < y_1$, then the paths in Graph(w) starting in $(y, 0)$ and in $(y_1, 0)$ would cross, which is not possible since no letter of Σ allows such a crossing. Repeating this process yields a strictly increasing sequence $y, y_1, \ldots, y_k$ in $[0, 2n-1]$, of length at most $2n$, such that the path in Graph(w^{2n}) starting in $(y, 0)$ contains the points $(y, 0), (y_1, |w|), \ldots (y_k, k|w|)$, and is cut before reaching $(k+1)|w|$ on the second component.

We can now use this lemma to prove that K_n is LTL-definable. We use the characterization of LTL-definable languages as aperiodic languages [7]:
A language L is LTL-definable if and only if there exists $N_0 \in \mathbb{N}$ such that for all $N \geq N_0$ and $u_1, u_2, u_3, v \in \Sigma^*$, we have both of the following :

$$(u_1 v^N u_2) u_3^\omega \in L \Leftrightarrow (u_1 v^{N+1} u_2) u_3^\omega \in L$$
$$u_1 (u_2 v^N u_3)^\omega \in L \Leftrightarrow u_1 (u_2 v^{N+1} u_3)^\omega \in L$$

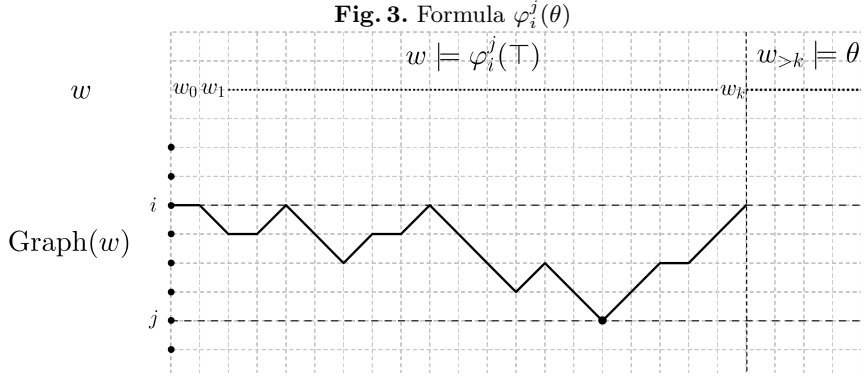
The fact that K_n satisfies this characterization is a consequence of Lemma 22 : when $N \geq 2n$, the uncut paths in the graphs Graph(v^N) and Graph(v^{N+1}) are exactly the paths starting in $(x, 0)$ for $x \in X_v$, and these paths end in x. Hence the existence of an infinite path in the graph of a word with v^N as a subword does not depend on the exponent on N as long as $N \geq 2n$.

B.2 An explicit LTL formula for K_n

We now give an explicit formula for the language K_n, using intermediate formulas with the following intuition:

- for $i < j$ and a formula θ, $\varphi_i^j(\theta)$ is a formula whose models are words that have an associated graph with a path starting in i, staying between i and j, touching j, and such that θ is true at the first point where the path comes back to i after touching j (if ever). Figure 3 gives a graphical representation of such a path.

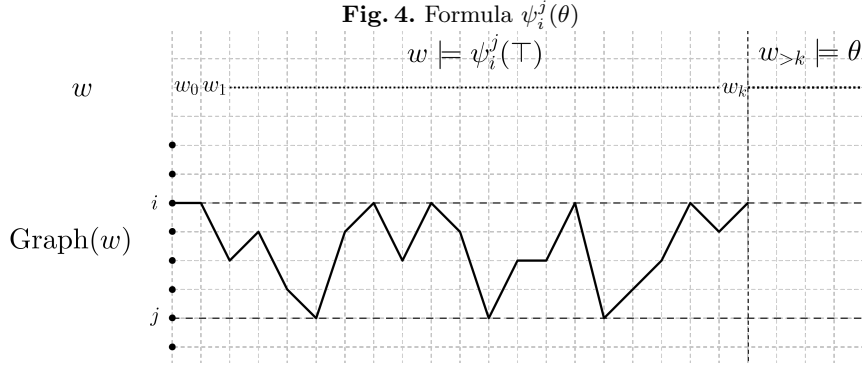
Fig. 3. Formula $\varphi_i^j(\theta)$



- for $i \leq j$ and a formula θ, $\psi_i^j(\theta)$ follows the same intuition as φ_i^j, except that the path of interest is allowed to travel several (possibly infinitely many) times betwen i and j before ending up in i and verifying θ; this is illustrated in Figure 4.

The idea behind the general formula is that if there is an infinite path in Graph(w) oscillating infinitely often between i and j, then it oscillates between i

Fig. 4. Formula $\psi_i^j(\theta)$



and at most $j-1$ until it goes from i to j (without coming back to i), at which point it oscillates between $i+1$ and j until it goes from j to i, and so on. This behavior will be expressible using our formulas φ and ψ.

We will define our formulas by recursion on $j-i$. Let $N = 2n-1$. For any $i \leq j$, we will use the following notation: $\psi_i^{\leq j}(\theta) := \bigvee_{i \leq k \leq j} \psi_i^k(\theta)$.

We start by defining for each i the subalphabet α_i that maps i to i, and the formula $\psi_i^i(\theta)$.

$$\begin{aligned} \alpha_i &= \quad \Sigma \setminus \{a_{i-1}, a_i, b_i, b_{i+1}\} \\ \psi_i^i(\theta) &= \alpha_i \mathbf{WU} \theta \end{aligned}$$

Assume now that the formulas $\psi_{i'}^{j'}$ have been defined for all $i' \leq j'$ such that $j'-i' < j-i$. We define the formulas φ_i^j and ψ_i^j by:

$$\begin{aligned} \varphi_i^j(\theta) &= \psi_i^{\leq j-1}(a_i \wedge \odot(\psi_{i+1}^j(b_i \wedge \odot\theta))) \\ \psi_i^j(\theta) &= \varphi_i^j(\top) \wedge [(\varphi_i^j(\top) \to \varphi_i^j(\varphi_i^j(\top)))\mathbf{WU}\varphi_i^j(\theta)] \end{aligned}$$

The language K_n is then described by the formula $\phi = \mathbf{F}(\bigvee_{0 \leq i \leq j \leq N} \psi_i^j(\bot))$.

Lemma 23. $[\![\phi]\!] = K_n$

Proof. We use the notation $w_{>r}$ for a word w to denote the suffix of w obtained by removing the first r letters of w, and $w_{\leq r}$ for the removed prefix. By a path "staying between i and j", we mean a path that contains only vertices of the form (k, r) for $i \leq k \leq j$.

We will prove inductively on $j-i$ the following induction hypothesis, for any formula θ :

1. for $i < j$, $w \models \varphi_i^j(\theta)$ if and only if there is a path in Graph(w) starting in $(i, 0)$, staying between i and j, which is either infinite, in which case it never comes back to i once it has touched j (if at all), or contains the vertex (j, r) for some $r > 0$, ending in a vertex (i, s) with $s > r$ and s minimal for this property, such that $w_{>s} \models \theta$.

2. for $i \leq j$, $w \models \psi_i^j(\theta)$ if and only if there is a path in Graph(w) starting in $(i,0)$, staying between i and j, which is either infinite, or contains the vertex (j,r) for some $r \geq 0$, and ends in a vertex (i,s) for some $s > r$ such that $w_{>s} \models \theta$.

The initial case is $i = j$. Let θ be a fixed formula. There is nothing to prove in point 1. In point 2, by the semantic of the **WU** operator, $w \models \psi_i^i(\theta)$ if and only if $w = uv$ with $u \in \alpha_i^*$, and either $v \in \alpha_i^\omega$, or $v \models \theta$. In the first case, the path starting in $(i,0)$ in Graph(w) is infinite ; in the second case, the rest of the induction hypothesis is satisfied, by choosing $r = 0$ and s such that $v = w_{>s}$.

Let us prove point 1 above for some $i < j$, assuming that the induction hypothesis has been proved for all i', j' such that $j' - i' < j - i$ and all formulas θ. We will prove point 2 afterwards, using point 1. Again, let θ be a fixed formula.

For the left to right implication, let $w \models \varphi_i^j(\theta)$. Then, there is $k < j$ such that $w \models \psi_i^k(a_i \wedge \odot(\psi_{i+1}^j(b_i \wedge \odot\theta)))$. By the induction hypothesis, there is in Graph(w) a path starting in $(i,0)$, staying between i and k, which is either infinite or contains the vertex (k,r) for some $r > 0$, a vertex (i,s) with $s > r$ and s minimal for this property, such that $w_{>s} \models a_i \wedge \odot(\psi_{i+1}^j(b_i \wedge \odot\theta))$. In the case where the path is infinite, the induction hypothesis is proven. In the other case, we know that $w_{>s} = a_i v$ with $v \models \psi_{i+1}^j(b_i \wedge \odot\theta)$, and using again the induction hypothesis and assuming that the path in Graph(v) starting in $(i+1,0)$ is not infinite, we know that $v = ub_i u'$ with Graph(u) having a path from $(i+1,0)$ to $(i+1,|u|)$ staying between $i+1$ and j and containing some vertex (j,r), and $u' \models \theta$. Putting all of this together, $w = w_{\leq r} a_i u b_i u'$, and we see that Graph$(w)$ has a path starting in $(i,0)$, passing through some (j,r) for $r > 0$, and which stops at the first vertex (i,s) with $s > r$, and we have that $w_{>s} = u' \models \theta$, which concludes the first implication.

Conversely, let w be a word satisfying the right part of the "if and only if".

First case : If the path starting in $(i,0)$ stays between i and j and is infinite and never comes back to i once it touched j (if at all), then : either it never reaches j, in which case by the induction hypothesis $w \models \psi_i^{j-1}(\theta')$ for any formula θ', and in particular $w \models \varphi_i^j(\theta)$, and we are done ; or it passes through (j,r) for some r minimal. In this case, we can write $w = ua_i v$ with Graph(u) having a path starting in $(i,0)$, staying between i and k for some $k < j$, reaching k at some point, and ending in $(i,|u|)$, and with the path in Graph(v) starting in $(i+1,0)$ staying between $i+1$ and j (since it cannot come back to i). Hence, $v \models \psi_{i+1}^j(\theta')$ for any θ' by the induction hypothesis, and again by the induction hypothesis on $\psi_i^k(a_i \wedge \odot\psi_{i+1}^j(\theta'))$ for any θ', we obtain that $w \models \varphi_i^j(\theta)$.

Second case : If the path starting in $(i,0)$ stays between i and j and contains the vertex (j,r) for some $r > 0$, ends in a vertex (i,s) with $s > r$ and s minimal for this property, such that $w_{>s} \models \theta$, then by a similar approach, we have that $w = ua_i v b_i w_{>s}$ with : the path in Graph(u) starting in $(i,0)$ staying between i and k for some $k < j$, touching k at some point, and ending in $(i,|u|)$; and the path in Graph(v) starting in $(i+1,0)$ staying between $i+1$ and j, touching j at some point, and ending in $(i+1,|v|)$. Hence, by the induction hypothesis, we see

that $vb_iw_{>s} \models \psi^j_{i+1}(b_i \wedge \odot\theta)$, and that $w \models \varphi^j_i(\theta)$.

We now prove point 2 above. Again, fix some formula θ.

For the left to right implication, let $w \models \psi^j_i(\theta)$. We first note that $w \models \varphi^j_i(\top)$. By the induction hypothesis, there are two cases :

First case : The path in Graph(w) starting in $(i, 0)$ stays between i and j and is infinite. In this case, the induction hypothesis is satisfied.

Second case : The path in Graph(w) starting in $(i, 0)$ stays between i and j, and comes back to i after touching j. In this case, we will need to inspect the second part of the formula. Since $w \models (\varphi^j_i(\top) \to \varphi^j_i(\varphi^j_i(\top)))\mathbf{WU}\varphi^j_i(\theta)$, the formula $\varphi^j_i(\top) \to \varphi^j_i(\varphi^j_i(\top))$ must be true of all suffixes of w, or of all suffixes of the form $w_{>s}$ for $s < t$ for some fixed t, and $w_{>t} \models \varphi^j_i(\theta)$. In the first case, by the induction hypothesis, for every s, if $w_{>s} \models \varphi^j_i(\top)$, then there is a path in Graph($w_{>s}$) starting in $(i, 0)$, reaching (j, r) for some $r > 0$, coming back to (i, s'), and $w_{>s'} \models \varphi^j_i(\top)$. Consequently, the path starting in $(i, 0)$ in Graph(w) is infinite. In the second case, the same phenomenon occurs until the time t. If the path passes through (i, t), then since $w_{>t} \models \varphi^j_i(\theta)$, the path must continue and either be infinite (in which case we are done), or reach j then come back to (i, t'), and $w_{>t'} \models \theta$, and we are also done. The only case left to consider is the case where the path passes through (k, t) for some $k > i$. Since $w_{>t} \models \varphi^j_i(\theta)$, the path in Graph($w_{>t}$) starting in $(i, 0)$ cannot reach j since by doing so it would cut the original path, thus contradicting the induction hypothesis ; hence, it must be infinite, and in this case the original path cannot come back to i since it would cut the new path, so it is forced to be infinite by induction hypothesis, and we are done.

Finally, let us consider the right to left implication. Consider w satisfying the right side of the "if and only if". We again distinguish two cases :

First case : We can write $w = u_1u_2 \dots u_pv$ for some $p \geq 1$ with : for all $l \in [1, p]$, Graph(u_l) has a path starting in $(i, 0)$, staying between i and j, reaching (j, r) for some $r > 0$, and coming back to i for the first time after r only in $(i, |u_l|)$; and $v \models \theta$. In this case, $u_pv \models \varphi^j_i(\theta)$. If $p = 1$, then since in particular $u_pv \models \varphi^j_i(\top)$, we have $w = u_pv = \psi^j_i(\theta)$. If $p > 1$, we prove by recurrence that for all $l \in [1, p]$, $u_lu_{l+1} \dots u_pv \models (\varphi^j_i(\top) \to \varphi^j_i(\varphi^j_i(\top)))\mathbf{WU}\varphi^j_i(\theta)$. This is true for $l = p$ (since in this case, the left part of the **WU** is not relevant). If this is true for some $l + 1 \leq p$, then let us prove it for l. It is enough to prove that for all $r \in [0, |u_l|]$, $(u_l)_{>r}u_{l+1} \dots u_pv \models \varphi^j_i(\top) \to \varphi^j_i(\varphi^j_i(\top))$. If the path in Graph($u_l$) starting in $(i, 0)$ reaches (i, r), then $(u_l)_{>r}u_{l+1} \dots u_pv \models \varphi^j_i(\top)$ by assumption on u_l. In this case, by assumption on u_{l+1}, we also have $(u_l)_{>r}u_{l+1} \dots u_pv \models \varphi^j_i(\varphi^j_i(\top))$, and the property follows. If the path in Graph(u_l) starting in $(i, 0)$ reaches (k, r) for some $k > i$, then we claim that $(u_l)_{>r}u_{l+1} \dots u_pv \not\models \varphi^j_i(\top)$; for if this was the case, then by induction hypothesis there would be a path starting in (i, r), staying between i and j, and being either infinite or reaching j at some point, contradicting the fact that the path in (k, r) must come back to i and path-crossing is not possible. Hence, the property is proved. Finally, since $p \geq 1$, we see that $u_1 \dots u_pv \models \varphi^j_i(\top)$, and hence $w \models \psi^j_i(\theta)$.

Second case : If the path in Graph(w) starting in $(i,0)$ stays between i and j and is infinite, then there are two possible subcases. The first subcase is the one where the path oscillates infinitely often between i and j, we can write $w = u_1 u_2 \dots u_p \dots$ with the words u_p having the same property as in the first case above. But in this case, by a reasoning similar to the one in the first case, we see that $w \models \varphi_i^j(\top)$ and $w \models \mathbf{G}(\varphi_i^j(\top) \to \varphi_i^j(\varphi_i^j(\top)))$, and hence $w \models \psi_i^j(\theta)$. The second subcase is the one where the path oscillates between i and j for some time before leaving i and never reaching j, or leaving j and never reaching i. In this case, we can write $w = u_1 \dots u_p v$ as above, with $p \geq 0$ (there might be no u_l at all) and the u_l satisfying the same conditions as above, and v being such that the path in Graph(v) starting in $(i,0)$ stays between i and j, and either never reaches j, or reaches j but never reaches i again afterwards. By the induction hypothesis, $v \models \varphi_i^j(\theta)$, and a reasoning similar to the first case above (starting with v instead of $u_p v$) leads to the same conclusion.

To conclude the proof, we now only have to show that the models of ϕ are exactly the words from K_n. If $w \in K_n$, then there is s such that $w_{>s}$ has an infinite path starting in $(i,0)$ and staying between i and j for some values of i and j. Hence, by our point 2 above, $w_{>s} \models \psi_i^j(\bot)$, and so $w \models \phi$. Conversely, if $w \models \phi$, then there is s such that $w_{>s} \models \psi_i^j(\bot)$. But by our result above, there is a path in Graph($w_{>s}$) starting in $(i,0)$ that stays between i and j and which is infinite, since $\bot$ can never be satisfied. Consequently, $w \in K_n$, and this concludes the proof.

B.3 Size of the formula

Lemma 24. *We have* $2^{N-1} < |\phi|_{dag} < 3^{N+1}(N+4)!(5N+4)$*, where* $N = 2n-1$.

Proof. Upper bound

For $0 \leq k \leq N$, let $A(k) = 3^k(k+1)!(5k+4)N$ and $B(k) = 3A(k)+4$. We will prove inductively on $k = j - i$ that

$$|\psi_i^j(\theta)|_{dag} \leq B(k) + |\theta|_{dag} \quad \text{for } k \geq 0$$
$$|\varphi_i^j(\theta)|_{dag} \leq A(k) + |\theta|_{dag} \quad \text{for } k \geq 1$$

For the case $k = 0$, note that $|\psi_i^i(\theta)|_{dag} = |\alpha_i \mathbf{W}\mathbf{U}\theta|_{dag} < 2N + |\theta|_{dag} < B(0) + |\theta|_{dag}$.

For the induction case, assume that the formulas are true for $i - j < k$, and take i and j such that $j - i = k$. Note that in the computation below, we make use of the fact that a subformula that appears at different places in a formula needs to be counted only once in the DAG-size; the places where this fact has been used are marked with a $(*)$.

$$\begin{aligned}
|\varphi_i^j(\theta)|_{dag} &= |\psi_i^{\leq j-1}(a_i \wedge \odot(\psi_{i+1}^j(b_i \wedge \odot\theta)))|_{dag} \\
&\leq (k-2) + (k-1)B(k-1) + |a_i \wedge \odot(\psi_{i+1}^j(b_i \wedge \odot\theta))|_{dag} \qquad (*) \\
&\leq (k-2) + (k-1)B(k-1) + 3 + B(k-1) + 3 + |\theta|_{dag} \\
&\leq k + 4 + kB(k-1) + |\theta|_{dag} \\
&\leq k + 4 + 3kA(k-1) + 4k + |\theta|_{dag} \\
&\leq 3kA(k-1) + 5k + 4 + |\theta|_{dag} \\
&\leq 3k(3^{k-1}k!(5(k-1)+4)N + 5k + 4 + |\theta|_{dag} \\
&\leq 3^k.k.k!(5k-1)N + 5k + 4 + |\theta|_{dag}
\end{aligned}$$

Now, it is easy to check that if $N \geq \frac{1}{2}$, then this last expression is less than $A(k) + |\theta|_{dag}$, which concludes the induction step for $|\varphi_i^j(\theta)|_{dag}$.

$$\begin{aligned}
|\psi_i^j(\theta)|_{dag} &= |\varphi_i^j(\top) \wedge [(\varphi_i^j(\top) \rightarrow \varphi_i^j(\varphi_i^j(\top)))\mathbf{WU}\varphi_i^j(\theta)]|_{dag} \\
&\leq 3 + A(k) + 1 + A(k) + A(k) + |\theta|_{dag} \qquad (*) \\
&\leq 3A(k) + 4 + |\theta|_{dag}
\end{aligned}$$

This concludes the induction proof. We now prove the upper bound on ϕ (using that $(5N+4)(N+1) \geq (5N+4)N + 6$ when $N \geq 1$):

$$\begin{aligned}
|\phi|_{dag} &= |\mathbf{F}(\textstyle\bigvee_{0\leq i\leq j\leq N} \psi_i^j(\bot))|_{dag} \\
&\leq 1 + (N+1)^2 - 1 + (N+1)^2(B(N)+1) \ (*) \\
&\leq (N+1)^2(3A(N)+6) \\
&\leq (N+1)^2(3^{N+1}(N+1)!(5N+4)N + 6) \\
&\leq (N+1)^2(3^{N+1}(N+1)!(5N+4)(N+1) \\
&\leq 3^{N+1}(N+4)!(5N+4)
\end{aligned}$$

Lower bound

We will make use of the following easy fact:

Fact 25 *For all formulas φ, $|\varphi|_{dag}$ is greater than or equal to the depth of the syntactic tree of φ.*

We will actually prove that for any formula θ, $\psi_0^N(\theta)$ has a syntactic tree of depth at least 2^{N-1}, which is enough for our purpose by Fact 25 since $\psi_0^N(\bot)$ is a subformula of ϕ.

In order to prove this result for any θ, we consider θ as a free variable, and we compute the maximal depth at which appears this variable in the formula $\psi_i^j(\theta)$ for $i \leq j$. We first note that for a fixed value of $k = j - i$, all the formulas $\psi_i^j(\theta)$ have the same structure, the only difference being a renaming of the leaves of the syntactic tree ; for this reason, we will call d_k the maximal depth of the variable θ in these formulas. Now, assume that $i < j$; we will follow a branch of the syntactic tree of $\psi_i^j(\theta)$. Consider the subformula $\varphi_i^j(\theta)$ of $\psi_i^j(\theta)$. In this subformula, we consider the branch that goes through $\psi_i^{j-1}(a_i \wedge \odot(\psi_{i+1}^j(b_i \wedge \odot\theta)))$ at the outermost $\bigvee_{i\leq k\leq j-1}$ operator. In this subformula, we again follow the path passing through the subformula $\psi_{i+1}^j(b_i \wedge \odot\theta)$, then the path to the variable θ.

Summing up, we have encountered along this path the formulas $\psi_i^{j-1}(\theta_1)$ and $\psi_{i+1}^{j}(\theta_2)$ for some formulas θ_1 and θ_2, one nested in the other, before reaching the leaf θ. Hence, $d_{j-i} > 2d_{j-i-1}$. By recursion, we get that $d_{n-1} > 2^{n-1}$, which yields the announced minoration of $|\phi|_{dag}$.

C Properties of *ESafe*

C.1 Proof of Lemma 10

Proof. First of all, recall that it is equivalent whether L is DCW or NCW recognizable [17], and that it is decidable whether it is the case [3].

Now, if L is given by a DCW $\mathcal{D}$, by Theorem 9, the problem amounts to checking whether L is prefix-independent. It is the case if and only if all reachable states of $\mathcal{D}$ accept the same language. This can be verified in NL, by guessing two nonequivalent reachable states, and a lasso in $\mathcal{D} \times \mathcal{D}$ that is accepting for one but not for the other. This is actually a NL procedure for the complement, but since NL is closed under complement, we obtain that membership in *ESafe* is in NL for DCW inputs.

If L is given by a NCW $\mathcal{C}$, the PSPACE-hardness can be obtained by reduction from NFA universality, which is PSPACE-complete. To reduce it to this problem, consider an arbitrary NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ on Σ. Let $\$ \notin \Sigma$ be a new letter, and $q_f \notin Q$ be a new state. We build a NCW $\mathcal{C} = (Q', \Sigma', \Delta', q_0, \{q_f\})$ by setting $\Sigma' = \Sigma \uplus \{\$\}$, $Q' = Q \uplus \{q_f\}$ and $\Delta' = \Delta \cup \{(q, \$, q_f) \mid q \in F \cup \{q_f\}\}$. We have $L(\mathcal{A}') = L(\mathcal{A})\$^\omega$, and $L(\mathcal{A}') \in \mathit{ESafe}$ if and only if $L(\mathcal{A}) = \Sigma^*$. This shows that the problem is PSPACE-hard.

The membership of the problem in PSPACE directly follows from the fact that the problem is in NL for DCW. Indeed, we can use this algorithm with the determinization of $\mathcal{C}$ obtained via breakpoint construction [17], yielding a PSPACE procedure.

C.2 Proof of Theorem 7

Recall that Theorem 7 states that a language is definable in $S\nu$TL if and only if it is recognized by a safety automaton.

Proof. Recall that safety languages are exactly those accepted by safety automata (i.e. with all states accepting), regardless of whether they are deterministic, nondeterministic or alternating. Indeed, removing alternation or determinizing can be done while preserving the safety condition, through simple powerset constructions.

We show that any $S\nu$TL formula can be turned to an alternating safety automaton, as a particular case of the general construction for branching μ-calculus [20].

Let ψ be a $S\nu$TL formula, and $sub(\psi)$ be the set of subformulas of ψ. We build the alternating safety automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ with ϵ-transitions, by choosing as set of states $Q = \{q_\varphi \mid \varphi \in sub(\psi)\} \cup \{\top\}$, and $q_0 = q_\psi$.

The transition function δ is defined as follows, where a ranges over Σ.

- $\delta(q_a, a) = \delta(\top, a) = \top$
- $\delta(q_{\varphi_1 \vee \varphi_2}, \epsilon) = q_{\varphi_1} \vee q_{\varphi_2}$
- $\delta(q_{\varphi_1 \wedge \varphi_2}, \epsilon) = q_{\varphi_1} \wedge q_{\varphi_2}$
- $\delta(q_{\odot\varphi}, a) = q_\varphi$
- $\delta(q_X, \epsilon) = q_{\varphi_X}$, where $\nu X.\varphi_X$ is the formula bounding the variable X in ψ.
- $\delta(q_{\nu X.\varphi}, \epsilon) = q_\varphi$.

The values not specified here mean that the function is undefined on these values, for instance $\delta(a, b)$ with $a \neq b$.

It is shown in [20] for general branching μ-calculus that $L(\mathcal{A}) = [\![\psi]\!]$.

Conversely, acceptance of a deterministic safety automaton can be expressed by a $S\nu$TL formula.

Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, Q)$ be such an automaton, with $Q = \{q_0, \ldots, q_n\}$. We define formulas φ_i and ψ_i recursively for i from n to 0. If ψ_j is defined for all $j > i$:

$$\varphi_i = \left(\bigvee_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_j \\ j > i}} (a \wedge \odot \psi_j) \right) \vee \left(\bigvee_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_j \\ j \leq i}} (a \wedge \odot X_j) \right) \text{ and } \psi_i = \nu X_i.\varphi_i$$

It is clear that ψ_0 is a closed formula, and we claim that $L(\psi_0) = L(\mathcal{A})$.

This can be proved recursively with the following induction hypothesis at step i : for the interpretation of the variables defined by $val(X_k) = L(\mathcal{A}, q_k)$ for all k, we have that $L(\mathcal{A}, q_i) = [\![\psi_i]\!]_{\mu, val}$. Assuming that this hypothesis is satisfied for all $j > i$, we consider the following equation satisfied by $L(\mathcal{A}, q_i)$:

$$S = \left(\bigcup_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_j \\ j > i}} aL(A, q_j) \right) \cup \left(\bigcup_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_j \\ j < i}} aL(A, q_j) \right) \cup \left(\bigcup_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_i}} aS \right)$$

Using the induction hypothesis for the $j > i$ and the valuation val for the $j < i$, the following equation is again satisfied by $L(\mathcal{A}, q_i)$:

$$S = \left(\bigcup_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_j \\ j > i}} a[\![\psi_j]\!]_{\mu, val} \right) \cup \left(\bigcup_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_j \\ j < i}} a val(X_j) \right) \cup \left(\bigcup_{\substack{a \in \Sigma \\ \delta(q_i, a) = q_i}} aS \right)$$

In other words, $L(\mathcal{A}, q_i)$ is a fixed point for $S \mapsto [\![\varphi_i]\!]_{\mu, val[X_i \to S]}$, hence by the semantic of the νX_i operator, $L(\mathcal{A}, q_i) \subseteq [\![\psi_i]\!]_{\mu, val}$. Moreover, if S is

a fixed point, then it satisfies the two equations above (using the induction hypothesis for the first one). If $w \in S$, then either w belongs to one of the first two terms of the union in the first equation, in which case it is clearly in $L(\mathcal{A}, q_i)$, or it belongs to the last term, in which case it is of the form $w = av$ with $a \in \Sigma$, $\delta(q_i, a) = q_i$, and v in S. Repeating this process on v, we see that $w \in L(\mathcal{A}, q_i)$, so $S \subseteq L(\mathcal{A}, q_i)$, and $L(\mathcal{A}, q_i)$ is the greatest fixed point of $S \mapsto [\![\phi_i]\!]_{\mu, val[X_i \to S]}$, that is to say $L(\mathcal{A}, q_i) = [\![\psi_i]\!]_{\mu, val}$, and the induction hypothesis at step i is proven. Finally, step 0 of the induction hypothesis states exactly that $[\![\psi_0]\!]_{\mu, val} = L(\psi_0) = L(\mathcal{A}, q_0) = L(\mathcal{A})$.

C.3 Proof of Theorem 9

We recall the statement of Theorem 9:

Theorem 9. *ESafe is equal to the class of prefix-independent coBüchi languages. Moreover, if $L = \Sigma^* L_{safe}$ is accepted by a NCW $\mathcal{C}$, we can build a non-deterministic safety automaton $\mathcal{A}_{safe}$ from $\mathcal{C}$ recognizing L_{safe} with the same number of states. Conversely, if we have a non-deterministic safety automaton for L_{safe}, we can build a NCW $\mathcal{C}$ for L with one more state.*

We first prove the right to left inclusion. Let L be a prefix-independent coBüchi language, recognized by a NCW $\mathcal{C} = (Q, \Sigma, \Delta, q_0, F)$, where all states are reachable from q_0. We define the safety automaton $\mathcal{A}_{safe} = (Q_{safe}, \Sigma, \Delta_{safe}, Q_{safe}, Q_{safe})$ with $Q_{safe} = F$, $\Delta_{safe} = \Delta \cap (Q_{safe} \times \Sigma \times Q_{safe})$; $\mathcal{A}_{safe}$ is the safety automaton obtained from $\mathcal{C}$ by suppressing all the non-accepting states, and taking all accepting states as initial. Notice that $L(\mathcal{A}_{safe})$ is suffix-closed, since all states of $\mathcal{A}_{safe}$ are also initial states. Finally, let $\mathcal{A}'$ be the coBüchi automaton with ϵ-transitions defined by $(Q', \Sigma, \Delta', q'_0, \{q'_0\})$, with $Q' = Q_{safe} \cup \{q'_0\}$, $\Delta' = \delta_{safe} \cup \{(q'_0, a, q'_0) \mid a \in \Sigma\} \cup \{(q'_0, \epsilon, Q_0)\}$: for a word to be accepted by this automaton, we must read any prefix of the word, then jump to any initial state of $\mathcal{A}_{safe}$, from which we must stay in $\mathcal{A}_{safe}$. It follows that $L(\mathcal{A}') = \Sigma^* L(A_{safe})$, and $L(A_{safe})$ is a suffix-closed safety language, so $L(\mathcal{A}') \in \mathit{ESafe}$.

We claim that $L(\mathcal{A}') = L(\mathcal{C})$, which will prove that $L(\mathcal{C}) \in \mathit{ESafe}$. First, if $w \in L(\mathcal{C})$, then by definition of the coBüchi acceptance condition, $w = uv$ with $u \in \Sigma^*$, and $v \in L(\mathcal{A}_{safe})$, so $w \in \Sigma^* L(\mathcal{A}_{safe}) = L(\mathcal{A}')$. Conversely, consider $w \in L(\mathcal{A}')$. Then, the accepting run of $\mathcal{A}'$ on w can be decomposed into $q'_0 \xrightarrow{u} q'_0 \xrightarrow{\epsilon} q \xrightarrow{v} \dots$, where v is accepted in $\mathcal{A}'$ from q. This implies that v is accepted in $\mathcal{C}$ from q. Since q is an accessible state in $\mathcal{C}$ there is $u' \in \Sigma^*$ such that $q_0 \xrightarrow{u'} q$ in $\mathcal{C}$, and we get $u'v \in L(\mathcal{C})$. Since $L(\mathcal{C})$ is prefix-independent, we also have that $uv = w \in L(\mathcal{C})$, which concludes the proof of the equality $L(\mathcal{C}) = L(\mathcal{A}')$.

Let us now prove the left to right inclusion. Let $L \in \mathit{ESafe}$, i.e. $L = \Sigma^* L_{safe}$ with L_{safe} a suffix-closed safety language. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q)$ be a non-deterministic safety automaton for L_{safe}, with all states accessible from q_0. As before, we build a NCW $\mathcal{C}$ for L by adding to $\mathcal{A}$ a non-accepting initial state q'_0, with transitions $q'_0 \xrightarrow{a} q'_0$ for all $a \in \Sigma$, and $q'_0 \xrightarrow{\epsilon} q$ for all $q \in Q$. This shows

that L is a coBüchi language. Moreover, L_{safe} is suffix closed, so for all $u, v \in \Sigma^*$ and $w \in \Sigma^\omega$, we have $uw \in \Sigma^* L_{safe} \Leftrightarrow \mathit{suff}(w) \cap L_{safe} \neq \emptyset \Leftrightarrow vw \in \Sigma^* L_{safe}$, showing that L is prefix-independent.

D Constructions using $E\nu$TL

D.1 Proof of Lemma 11

We define inductively the formulas ψ_i for i from N to 0, each one containing X_{i-1} as a free variable, except for ψ_0 which is closed (recall that we use the alphabets $(\alpha_i)_{0 \leq i \leq N}$ defined in Section B.2) :

$$\begin{aligned} \psi_N &= \nu X_N.((\alpha_N \wedge \odot X_N) \vee (b_{N-1} \wedge \odot X_{N-1})) \\ \text{For } 0 < i < N : \psi_i &= \nu X_i.((\alpha_i \wedge \odot X_i) \vee (a_i \wedge \odot \psi_{i+1}) \vee (b_{i-1} \wedge \odot X_{i-1})) \\ \psi_0 &= \nu X_0.((\alpha_0 \wedge \odot X_0) \vee (a_0 \wedge \odot \psi_1)) \end{aligned}$$

We finally define $\Phi := \psi_0$.

Lemma 11. *The formula Φ has size linear in n, and $[\![\Phi]\!]_E = K_n$.*

Proof. Due to the definition of the eventual semantic, we only have to show that the usual μ-calculus semantic of Φ is the set of words that have an infinite path in Graph(w) starting in $(0, 0)$. For all i, let S_i be the set of words w such that Graph(w) has an infinite path starting in $(i, 0)$, and let val be the valuation defined by $val(X_i) = S_i$.

Let us prove by induction on i for i from N to 0 that $[\![\psi_i]\!]_{\mu,val} = val(X_i)$. Note that the case $i = 0$ is exactly what we want to prove the lemma.

For the case $i = N$, let $\varphi_N = (\alpha_N \wedge \odot X_N) \vee (b_{N-1} \wedge \odot X_{N-1})$, so that $\psi_N = \nu X_N.\varphi_N$. We have to show that S_N is the greatest fixed point of the map $S \mapsto [\![\varphi_N]\!]_{\mu,val[X_N \to S]}$. But $[\![\varphi_N]\!]_{\mu,val[X_N \to S]} = \alpha_N S \cup b_{N-1} val(X_{N-1})$. Since a word w such that Graph(w) has an infinite path starting in $(N, 0)$ either starts by an element of α_N followed by $w' \in S_N$, or by b_{N-1} followed by $w' \in S_{N-1}$, we see that S_N is a fixed point of $S \mapsto [\![\varphi_N]\!]_{\mu,val[X_N \to S]}$. Moreover, if S is another such fixed point, then S starts by an element of α_N followed by an element of S, or by b_{N-1} followed by an element of S_{N-1}, and by definition of the sets S_i, this implies that any element w of S belongs to S_N. Hence, S_N is the maximal fixed point of the above map, and we have that $S_N = [\![\psi_N]\!]_{\mu,val}$.

For the induction case (be it for $i > 0$ or $i = 0$), the proof is completely similar once we notice that by the induction hypothesis, $[\![\psi_{i+1}]\!]_{\mu,val} = S_{i+1}$.

D.2 A $E\nu$TL formula for the original L_n

We show here that the logic $E\nu$TL can also express in a succinct way the original family of languages (L_n) witnessing exponential succinctness of GFG automata. Let $N = 2n - 1$.

We recall that the language L_n on $\Sigma = \{\iota, \tau, \sigma, \sharp\}$ is defined by interpreting the letters as permutations on the set $[0, N]$ ($\sharp$ being partial) : ι is the identity,

$\tau = (0\ 1)$, $\sigma = (0\ 1 \dots N)$, $\sharp(0) = \bot$ and $\sharp|_{[1,N]} = id$. We define $G(w)$ similarly as above, and define L_n to be the set of words w such that $G(w)$ has an infinite path. We will define a formula φ such that $[\![\varphi]\!]$ defines the words w such that Graph(w) has an infinite path starting in $(0,0)$, and $[\![\varphi]\!]_E$ is L_n.

We define inductively (for i from N to 0) the formulas φ_i using the variables $X_0, \dots, X_N$ (with φ_i having as only free variable X_0, except for φ_0 which is closed) :

$$\varphi_N = \nu X_N.\big(\,((\tau \vee \sharp \vee \iota) \wedge \odot X_N) \vee (\sigma \wedge \odot X_0)\big)$$

$$\text{For } 1 < i < N : \varphi_i = \nu X_i.\big(\,((\tau \vee \sharp \vee \iota) \wedge \odot X_i) \vee (\sigma \wedge \odot \varphi_{i+1})\big)$$

And finally :

$$\varphi_1 = \nu X_1.\big((\tau \wedge \odot X_0) \vee ((\sharp \vee \iota) \wedge \odot X_1) \vee (\sigma \wedge \odot \varphi_2)\big)$$

$$\varphi_0 = \nu X_0.\big((\iota \wedge \odot X_0) \vee ((\tau \vee \sigma) \wedge \odot \varphi_1)\big)$$

Lemma 28. $[\![\varphi_0]\!]_E = L_n$.

Proof. The proof is essentially similar to the proof of Lemma 11, using the valuation such that $val(X_i)$ is the set of words w such that Graph(w) has an infinite path starting in $(i,0)$. For this reason, we do not detail it here.

D.3 Proof of Theorem 12

Theorem 12. *$\mathcal{C}$ is a GFG-NCW for $[\![\psi]\!]_E$.*

The proof that $\mathcal{C}$ recognizes $[\![\psi]\!]_E$ is similar to the proof of Lemma 16 in Appendix A.2, only in a more general context.

Proof. First, let us remark that $L(\mathcal{C}) = [\![\psi]\!]_E$. Indeed, Let w be a word accepted in $\mathcal{C}$. Then the accepting run has a suffix not using $\bot$, that is to say accepting from some state p in $\mathcal{D}_{\min}$. Since we assumed p is reachable, we get that $w = uv$ where $u'v \in L(\mathcal{A})$. We get that $\mathit{suff}(w) \cap \mathit{suff}(L(A_{\min})) \neq \emptyset$, and since $L(A_{\min}) = [\![\varphi]\!]$, by definition of the eventual semantic, we get $w \in [\![\psi]\!]_E$. Conversely, let w be a word in $[\![\psi]\!]_E$, then it has a suffix that is accepted by some run ρ from some q in $A_{\min}$. Upon reading this suffix, if $\mathcal{C}$ encounters $\bot$ it can jump to the current state of ρ and accept the remaining of the suffix. This concludes the proof that $L(\mathcal{C}) = [\![\psi]\!]_E$.

To prove that $\mathcal{C}$ is GFG, it is enough to construct a function $\sigma\colon \Sigma^* \to Q'$ that for every ω-word $w \in [\![\psi]\!]_E$ produces an accepting run of $\mathcal{C}$ over w. We will do it inductively with $\sigma(\epsilon) = q_0$.

Let σ follow deterministically the transitions of $\mathcal{C}$ for all the states $q \neq \bot$. It remains to define $\sigma(ua)$ if $\sigma(u) = \bot$ and a successive letter a is given. Assume that $|ua| = k$.

We now consider the run-DAG G of $\mathcal{C}$ on u. Vertices of G are from $Q' \times \mathbb{N}$. We consider that a path is *cut* in G every time it encounters $\bot$. Notice that since $w \in [\![\psi]\!]_E = L(\mathcal{C})$, G must contain an infinite uncut path.

For every $p \in Q$ let π_p be an uncut path of maximal length in G containing the node (p, k). Note that each of these paths π_p has a *starting position* $s(\pi_p) \in \mathbb{N}$.

Let $\sigma(ua) = p$ such that π_p is the longest among the paths $\{\pi_{p'} \mid p' \in Q\}$, i.e. a path π_p such that $s(\pi_p)$ is minimal. If there are two paths equally old, we move to that with smaller p for an arbitrary ordering on Q.

We need to prove that σ produces an accepting run ρ_σ of $\mathcal{C}$ over w.

Let t be the minimal starting point of an infinite path π in G. Assume the run ρ_σ is rejecting. This means that infinitely often, ρ_σ encounters $\bot$, and jumps on a state p with $s(\pi_p) \leq t$. This is absurd since this can only be done $|Q| - 1$ times, as after this number of cuts only the path π (or a path π' reaching the same state) has been uncut and satisfies $s(\pi) \leq t$. This concludes the proof that $\mathcal{C}$ is a GFG-NCW for $[\![\psi]\!]_E$.

Notice that we defined an infinite-memory GFG strategy σ, but it is possible to define a GFG strategy σ' using memory $2^{|Q|}$, by remembering a set of states M to visit next. When in $\bot$, the automaton jumps to a state from M and removes it from M. The set M is updated when reading letters according to the transition function, and it is reinitialized to Q when empty, this also guarantees that any word from $[\![\psi]\!]_E$ is accepted, since every path is eventually visited until an infinite path is found.

D.4 Alternative Construction for Theorem 12 using strongly connected components

We describe here another possible construction of a GFG-NCW automaton equivalent to a given $E\nu$TL formula. The construction is similar to the one given in section 5.5, except that we only determinize the strongly connected components of the intermediate automaton $\mathcal{A}_{\text{nd}}$.

Let us build $\mathcal{A}_{\text{nd}}$ the same way as in section 5.5. Recall that if the formula to translate is ψ, then $\mathcal{A}_{\text{nd}}$ is a safety non-deterministic automaton recognizing $[\![\psi]\!]$. Consider $w \in \mathit{suff}([\![\psi]\!])$. Then there is $u = vw \in [\![\psi]\!]$, and an accepting run of $\mathcal{A}_{\text{nd}}$ over u ; this run eventually reaches some strongly connected component with state space C of $\mathcal{A}_{\text{nd}}$ and never leaves it. In particular, for some suffix w' of w, the run restricted to w' reaches only states from C.

Let $C_1, \ldots, C_n$ be the set of states of the strongly connected components of $\mathcal{A}_{\text{nd}}$. We define for $1 \leq i \leq n$ a safety non-deterministic automaton $\mathcal{A}_i$ which is informally the automaton obtained from $\mathcal{A}_{\text{nd}}$ by keeping only the states in C_i. More formally, again omitting the accepting states since the automata are safety, if $\mathcal{A}_{\text{nd}} = (Q, \Sigma, q_0, \Delta)$, then $\mathcal{A}_i = (C_i, \Sigma, C_i, \Delta|_{C_i \times \Sigma})$; note that the $\mathcal{A}_i$ have all their states as initial states, but it is easy to build an equivalent automaton with only one more state, and only one initial state. Each automaton $\mathcal{A}_i$ accepts exactly those suffixes w of words $u \in [\![\psi]\!]$ for which there is an accepting run of $\mathcal{A}_{\text{nd}}$ over u ending in C_i, and for which the run restricted to w reaches only states from C_i. Since the semantic we are ultimately interested in is the eventual

semantic, we can restrict ourselves to considering these automata independently from each other.

The automata $\mathcal{A}_i$ can then be determinized and minimized as in Section 5.5, giving deterministic safety automata $\mathcal{A}_{i,min} = (Q'_i, \Sigma, q_i, \Delta_i)$. Once again, we can assume that the $\mathcal{A}_{i,min}$ are strongly connected, as keeping only there strongly connected components will not change the eventual semantic we are aiming at.

We can perform one last optimization step: if there are states $p \in \mathcal{A}_{i,min}$ and $q \in \mathcal{A}_{j,min}$ with $i \neq j$ such that $L(p) \subseteq L(q)$, then $\mathcal{A}_{i,min}$ can be removed from the list of components. Indeed, if a word w is accepted from $p' \in \mathcal{A}_{i,min}$, then the same word can be accepted from $q' \in \mathcal{A}_{j,min}$. This is because $\mathcal{A}_{i,min}$ is strongly connected, so there is a path $p \xrightarrow{u} p'$ for some word $u \in \Sigma^*$, so $uw \in L(p) \subseteq L(q)$, and therefore if q' is the state reached from q via u in $\mathcal{A}_{j,min}$, we obtain $w \in L(q')$. Thus, it is enough to keep components covering the possible residual safe languages.

Finally, we build the automaton $\mathcal{C} = (Q', \Sigma, \bot, \Delta', F)$, where $Q' = \left(\bigcup_{1 \leq i \leq n} Q'_i\right) \cup \{\bot\}$, $F = \bigcup_{1 \leq i \leq n} Q'_i$, and

$$\Delta' = \left(\bigcup_{1 \leq i \leq n} \Delta_i\right) \cup \{(p, a, \bot) \mid \forall q \in F, (p, a, q) \notin \bigcup_{1 \leq i \leq n} \Delta_i\} \cup \left(\{\bot\} \times \Sigma \times \bigcup_{1 \leq i \leq n} Q'_i\right)$$

The automaton $\mathcal{C}$ thus defined is then a GFG-NCW automaton for $[\![\psi]\!]_E$, as can be seen through a proof completely similar to the proof of Theorem 12.